

U.S. DEPARTMENT OF COMMERCE
National Technical Information Service

AD-A034 606

SPEECH UNDERSTANDING RESEARCH

STANFORD RESEARCH INSTITUTE
MENLO PARK, CALIFORNIA

OCTOBER 1976

024099

Final Technical Report

Covering the Period 15 October 1975 through 14 October 1976

SPEECH UNDERSTANDING RESEARCH

Edited by: DONALD E. WALKER

With Contributions by: WILLIAM H. PAXTON
GARY G. HENDRIX
BARBARA J. GROSZ
RICHARD E. FIKES

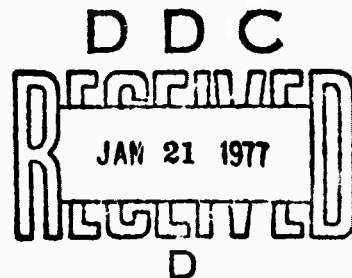
JONATHAN SLOCUM
ANN E. ROBINSON
JANE J. ROBINSON
DONALD E. WALKER

Prepared for:

DEFENSE ADVANCED RESEARCH PROJECTS AGENCY
ARLINGTON, VIRGINIA 22209

CONTRACT DAAG29-76-C-0011
ARPA Order No. 2903
Program Element Code 62706E

Approved for public release; distribution unlimited.



STANFORD RESEARCH INSTITUTE
Menlo Park, California 94025 • U.S.A.

REPRODUCED BY
**NATIONAL TECHNICAL
INFORMATION SERVICE**
U. S. DEPARTMENT OF COMMERCE
SPRINGFIELD, VA. 22161

STANFORD RESEARCH INSTITUTE
Menlo Park, California 94025

Approved for public release;
distribution unlimited.

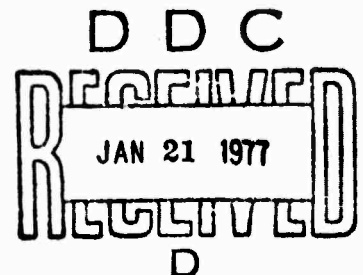
October 1976

Final Technical Report
Covering the Period 15 October 1975 through 14 October 1976
Stanford Research Institute Project 4762

SPEECH UNDERSTANDING RESEARCH

Edited By

Donald E. Walker
Project Leader
(415) 326-6200, Ext. 3071



With Contributions by

William H. Paxton, Gary G. Hendrix, Barbara J. Grosz, Richard E. Fikes,
Jonathan Slocum, Ann E. Robinson, Jane J. Robinson, and Donald E. Walker

CONTRACT DAAG29-76-C-0011
ARPA Order No. 2903
Program Element Code 62706E

Effective Date: 15 October 1975
Expiration Date: 14 October 1976
Amount of Contract: \$307,710

Prepared for

DEFENSE ADVANCED RESEARCH PROJECTS AGENCY
ARLINGTON, VIRGINIA 22209

The views and conclusions contained in this document are those of
the authors and should not be interpreted as necessarily
representing the official policies, either expressed or implied,
of the Defense Advanced Research Projects Agency or the
U.S. Government.

Approved by:

PETER E. HART, Director
Artificial Intelligence Center

EARLE D. JONES, Associate Executive Director
Information Science and Engineering Division

**BLANK PAGES
IN THIS
DOCUMENT
WERE NOT
FILMED**

ACCESSION for	
NTIS	White Section <input checked="" type="checkbox"/>
DDC	Soft Section <input type="checkbox"/>
UNANNOUNCED	<input type="checkbox"/>
JUSTIFIED	
BY	
DISTRIBUTION AND AVAILABILITY CODES	
DIS	SPECIAL
A	

ABSTRACT

This report is the final report in a series describing research performed by Stanford Research Institute over the past five years to develop the technology that will allow speech understanding systems to be designed and implemented for a variety of different task domains and environmental constraints.

Chapter I provides an overview of the speech understanding system we have developed, together with an example showing how an utterance is processed and some historical background. Chapters II and III present detailed descriptions of the definition system and the executive system that provide overall integration and control. Chapter IV discusses the results of experiments conducted to test alternative system control strategies. Chapters V, VI, and VII describe the representation of semantic knowledge, present a model of the problem domain, and show how semantic processing is used in the interpretation of an utterance. Chapters VIII, IX, and X deal with discourse and include discussions of dialog collection and analysis, the resolution of definite noun phrases, and ellipsis. Chapters XI, XII, and XIII indicate how the system responds to the interpreted utterance, how deduction is used both to find an answer and in the interpretation process, and how the system generates replies in English to a user. A final chapter lists publications and reports documenting the research we have performed on speech understanding during the past five years.

Preceding page blank

CONTENTS

LIST OF ILLUSTRATIONS	ix
I INTRODUCTION	I-1
A. ORIENTATION	I-1
B. AN OVERVIEW OF THE SPEECH UNDERSTANDING SYSTEM	I-4
C. AN EXAMPLE TO ILLUSTRATE PROCESSING IN THE SYSTEM	I-14
D. AN HISTORICAL PERSPECTIVE	I-25
II THE DEFINITION SYSTEM	II-1
A. INTRODUCTION	II-1
B. THE METALANGUAGE	II-2
C. A VERSION OF THE SRI LANGUAGE DEFINITION	II-15
D. THE DEFINITION COMPILER	II-27
E. DISCUSSION	II-42
III THE EXECUTIVE SYSTEM	III-1
A. INTRODUCTION	III-2
B. PARSE NET	III-4
C. OVERVIEW OF THE EXECUTIVE	III-7
D. DETAILS OF THE EXECUTIVE	III-20
E. DISCUSSION	III-75

Preceding page blank

IV	EXPERIMENTAL STUDIES	IV-1
A.	INTRODUCTION	IV-1
B.	EXPERIMENT 1 -- MAPPER PERFORMANCE	IV-2
C.	MAPPER SIMULATION	IV-5
D.	EXPERIMENT 2 -- FANOUT	IV-9
E.	EXPERIMENT 3 -- CONTROL STRATEGY DESIGN CHOICES . .	IV-13
F.	EXPERIMENT 4 -- GAPS AND OVERLAPS	IV-37
G.	EXPERIMENT 5 -- INCREASED VOCABULARY AND IMPROVED ACOUSTICS	IV-39
H.	DETAILED MEASUREMENTS OF EXECUTIVE OPERATION . . .	IV-44
I.	CONCLUSION	IV-53
J.	TEST SENTENCES	IV-54
V	THE REPRESENTATION OF SEMANTIC KNOWLEDGE	V-1
A.	INTRODUCTION	V-2
B.	THE ROLE OF SEMANTIC REPRESENTATION	V-4
C.	BASIC NETWORK NOTIONS	V-10
D.	PARTITIONING	V-20
E.	HIGHER-ORDER STRUCTURES	V-30
F.	AUGMENTATIONS	V-85
G.	SUPPORTS FOR DIVERSE TASKS	V-93
H.	LINEARIZED NET NOTATION	V-95
I.	APPLYING THE REPRESENTATION	V-99
VI	THE MODEL OF THE DOMAIN	VI-1

VII	SEMANTIC ASPECTS OF TRANSLATION	VII-1
A.	INTRODUCTION	VII-2
B.	PHASE I: SEMANTIC COMPOSITION	VII-5
C.	PHASE II: QUANTIFICATION	VII-36
D.	THE USE OF CASE INFORMATION	VII-55
VIII	DISCOURSE ANALYSIS	VIII-1
A.	INTRODUCTION	VIII-2
B.	DIALOG COLLECTION AND ANALYSIS	VIII-5
C.	COLLECTION OF THE DIALOGS	VIII-7
D.	ANALYSIS OF THE DIALOGS	VIII-15
IX	RESOLVING DEFINITE NOUN PHRASES	IX-1
A.	INTRODUCTION	IX-1
B.	THE FOCUS SPACE ENCODING OF CONTEXT	IX-9
C.	DEFNP RESOLUTION IN CONTEXT	IX-20
X	ELLIPSIS	X-1
A.	OVERVIEW	X-1
B.	SLOT DETERMINATION	X-5
C.	COMPLETING THE UTTERANCE	X-14
D.	ELLIPTICAL RELATIONAL NOUN PHRASES	X-26
E.	LIMITATIONS AND EXTENSIONS	X-29
XI	RESPONDING ON THE BASIS OF THE SEMANTIC TRANSLATION	XI-1
A.	PERSPECTIVE	XI-1
B.	INTERACTIONS WITH THE DEDUCTION COMPONENT AND THE ENGLISH GENERATOR	XI-3

XII	THE DEDUCTION COMPONENT	XII-1
A.	INTRODUCTION	XII-2
B.	ELEMENT PARITY	XII-10
C.	THE ENVIRONMENT TREE	XII-14
D.	THE EXECUTIVE FOR THE DEDUCTIVE COMPONENT	XII-18
E.	GENERATING CANDIDATE BINDINGS FOR A SELECTED QVISTA ELEMENT	XII-22
F.	RAMIFICATIONS OF A PROPOSED BINDING	XII-23
G.	THE BINDER	XII-26
H.	DERIVING ELEMENT-OF AND SUBSET RELATIONS USING TAXONOMIES	XII-31
I.	SIMPLIFICATION OF NEGATIONS	XII-33
J.	THE KVISTA EXTRACTOR	XII-34
K.	THE QVISTA EXTRACTOR	XII-41
L.	PROCEDURAL AUGMENTATION	XII-51
M.	TWO EXAMPLES	XII-58
XIII	GENERATING VERBAL RESPONSES	XIII-1
A.	INTRODUCTION	XIII-1
B.	GENERATION TEMPLATES	XIII-2
C.	NOUN PHRASES	XIII-7
D.	DISCUSSION	XIII-8
E.	LOCKING AHEAD	XIII-9
XIV	REFERENCES	XIV-1
XV	SRI SPEECH UNDERSTANDING RESEARCH PUBLICATIONS AND REPORTS	XV-1

ILLUSTRATIONS

I-1.	SRI CONTRIBUTIONS TO SPEECH UNDERSTANDING	I-4
I-2.	SYSTEM ORGANIZATION	I-5
II-1.	PART OF A COMPOSITION RULE	II-4
II-2.	SAMPLE LEXICAL ENTRY	II-8
II-3.	DECLARATIONS	II-11
II-4.	LEXICON	II-12
II-5.	RULES	II-14
II-6.	GLOBAL DECLARATIONS	II-17
II-7.	PHRASE STRUCTURE PARTS OF NUMBER RULES	II-19
II-8.	SMALLNUM RULE DEFINITION	II-21
II-9.	A PHRASE STRUCTURE DECLARATION AND ITS CORRESPONDING GRAPH	II-31
II-10.	NP GRAPH BEFORE ADDITION OF EXTRA NIL ARCS	II-34
III-1.	EXECUTIVE TASKS	III-9
III-2.	A CONSUMER PATH	III-13
III-3.	CREATE TERMINAL PHRASE	III-22
III-4.	DISTRIBUTE A PHRASE TO CONSUMERS	III-24
III-5.	ADD CONSTITUENT TO CONSUMER	III-26
III-6.	PART 1 OF PRELIMINARY ADD-CONSTITUENT TESTS	III-28
III-7.	PART 2 OF PRELIMINARY ADD-CONSTITUENT TESTS	III-30
III-8.	COMPLETE-PHRASE PROCEDURE	III-33
III-9.	CREATE AN INCOMPLETE PHRASE	III-38

III-11.	FILL-OUT-SUBNET PROCEDURE	III-44
III-10.	CREATE SUBNET PROCEDURE	III-42
III-12.	TRAVERSE SUBNET PROCEDURE	III-46
III-13.	NP-PREPP PARSE NET LOOP	III-47
III-14.	PASS 1 OF RATING ASSIGNMENT	III-50
III-15.	PASS 2 OF RATING ASSIGNMENT	III-51
IV-1.	MAPPER PERFORMANCE	IV-4
IV-2.	SCORE DISTRIBUTIONS FOR FALSE ALARMS AND HITS . .	IV-5
IV-3.	CUMULATIVE DISTRIBUTIONS OF HIT AND FALSE-ALARM SCORES	IV-6
IV-4.	FANOUT HISTOGRAM	IV-11
IV-5.	ACCURACY AND RUNTIME OF THE 16 DESIGNS	IV-16
IV-6.	CONTEXT CHECKING -- MAIN EFFECTS	IV-17
IV-7.	MAPPING ALL AT ONCE -- MAIN EFFECTS	IV-18
IV-8.	FOCUS BY INHIBITION -- MAIN EFFECTS	IV-19
IV-9.	ISLAND DRIVING -- MAIN EFFECTS	IV-20
IV-10.	MAIN EFFECTS OF VARIABLES ON PERCENT CORRECT . . .	IV-21
IV-11.	ACCURACY VERSUS LENGTH FOR ISLAND DRIVING	IV-22
IV-12.	FOCUS AND ISLAND-DRIVING INTERACTION	IV-22
IV-13.	STORAGE AND ACCURACY FOR THE 16 SYSTEMS	IV-24
IV-14.	MAIN EFFECTS ON STORAGE	IV-25
IV-15.	FOCUS AND ISLAND-DRIVING	IV-25
IV-16.	CONTEXT AND MAP-ALL INTERACTION	IV-26
IV-17.	MAIN EFFECTS ON TOTAL RUNTIME	IV-31
IV-18.	EFFECTS ON EXECUTIVE RUNTIME	IV-32
IV-19.	EFFECTS ON ACOUSTIC RUNTIME	IV-32
IV-20.	EFFECTS OF MODIFIED FOCUS BY INHIBITION	IV-37

IV-21.	EFFECTS OF GAP-OVERLAP PARAMETER	IV-38
IV-22.	OBSERVED DISTRIBUTION OF GAPS AND OVERLAPS	IV-39
IV-23.	ACCURACY RESULTS	IV-41
IV-24.	MAIN EFFECTS OF ACOUSTICS, VOCABULARY, AND MAP-ALL	IV-42
IV-25.	VOCABULARY AND MAP-ALL INTERACTION FOR ACOUSTIC RUNTIME	IV-42
IV-26.	ACOUSTICS AND MAP-ALL INTERACTION FOR FALSE TERMINALS	IV-43
IV-27.	COMPOSITION OF THE PARSE NET	IV-45
IV-28.	BLOCKING OF PHRASES AND PREDICTIONS	IV-46
IV-29.	EFFECTS OF LOOKAHEAD	IV-47
IV-30.	TIMING BREAKDOWN	IV-49
IV-31.	ACCURACY BREAKDOWN	IV-51
V-1.	FLOW OF SEMANTIC INFORMATION	V-5
V-2.	AN EXAMPLE SEMANTIC NETWORK	V-11
V-3.	ABSTRACTED USE OF DS ARCS	V-16
V-4.	ABSTRACTED USE OF DE ARCS	V-17
V-5.	THE USE OF DS AND DE ARCS	V-18
V-6.	SPACES SHOWING SYNTACTIC GROUPINGS	V-22
V-7.	ABSTRACTION OF VISTA ORDERING	V-24
V-8.	USE OF VISTA IN SYNTAX ENCODING	V-26
V-9.	EQUIVALENCE OF ENCLOSURE AND HEAVY ARROW NOTATION	V-27
V-10.	THE BELIEFS OF JOHN	V-29
V-11.	THE CONJUNCTION "THE HENRY.L.STIMSON WAS BUILT BY GENERAL.DYNAMICS AND THE HENRY.L.STIMSON IS OWNED BY THE U.S."	V-32
V-12.	THE DISJUNCTION "EITHER THE HENRY.L.STIMSON WAS BUILT BY GENERAL.DYNAMICS, OR THE HENRY.L.STIMSON IS OWNED BY THE U.S.S.R."	V-34

V-13.	THE HENRY.L.STIMSON IS OWNED BY EITHER THE U.S. OR THE U.S.S.R.	V-37
V-14.	THE U.S.S.R. DOES NOT OWN THE HENRY.L.STIMSON	V-38
V-15.	EITHER GENERAL.DYNAMICS DIDN'T BUILD THE HENRY.L.STIMSON OR THE U.S. OWNS IT	V-40
V-16.	IF GENERAL.DYNAMICS BUILT THE HENRY.L.STIMSON, THEN THE U.S. OWNS IT	V-41
V-17.	COMPACT IMPLICATION NOTATION	V-42
V-18.	A HIERARCHY OF QUANTIFICATION SPACES	V-48
V-19.	THE ENCODING OF $AxEy[p(x,y)]$ BY THE ORTHOGONAL PARTITION METHOD	V-51
V-20.	AN ENCODING OF $AxEy[p(x,y)]$ BY THE IMPLICIT EXISTENTIAL METHOD	V-55
V-21.	AN ENCODING OF $Ax[\{Ey[u(x,y)]\} \Rightarrow \{Ez[v(x,z)]\}]$. . .	V-58
V-22.	A SHORTHAND ENCODING OF $Ax[\{Ey[u(x,y)]\} \Rightarrow$ $\{Ez[v(x,z)]\}]$	V-59
V-23.	EVERY SUBMARINE IS OWNED BY SOME COUNTRY	V-62
V-24.	EVERY LAFAYETTE IS OWNED BY THE U.S.	V-63
V-25.	ALL THE SHIPS IN ANY GIVEN CLASS HAVE THE SAME LENGTH	V-65
V-26.	ALL SHIPS BUILT BY GENERAL.DYNAMICS BELONG TO THE U.S.	V-66
V-27.	THE DELINEATION THEOREM OF OWNINGS	V-69
V-28.	ABBREVIATED DELINEATION OF OWNINGS	V-71
V-29.	RELATING A SENTENCE TO ITS MEANING	V-72
V-30.	DID GENERAL.DYNAMICS BUILD THE HENRY.L.STIMSON? . . .	V-74
V-31.	DID GENERAL.DYNAMICS BUILD ALL U.S. DESTROYERS? . . .	V-75
V-32.	WHO BUILT THE HENRY.L.STIMSON?	V-77
V-33.	WHO BUILT EVERY DESTROYER?	V-80
V-34.	WHO BUILT EACH DESTROYER?	V-81
V-35.	WHAT COMPANIES BUILT WHAT DESTROYERS?	V-82

V-36.	HOW MANY SHIPS DID GENERAL.DYNAMICS BUILD?	V-84
V-37.	RELATING SUMS SITUATION TO FUNCTION PLUS	V-87
V-38.	AN APPLICATION OF SHIPDATA	V-89
V-39.	A KEYED-APPLICATIONS SITUATION	V-91
V-40.	THEOREM IMPLIED BY FIRST KEY	V-92
V-41.	NETWORK CREATED BY LN2	V-96
V-42.	AN LN2 STATEMENT	V-97
VI-1.	TOP OF DOMAIN MODEL	VI-2
VI-2.	DELINEATIONS OF MEASURES AND SPEEDS	VI-4
VI-3.	THE OVERLAPPING TAXONOMIES OF SHIPS AND SHIP.GROUPS .	VI-6
VI-4.	THE DELINEATION OF HAVE.BEAM	VI-10
VI-5.	ALL CVTS ARE TRAINING SHIPS	VI-11
VI-6.	LINKING HAVE.BEAM SITUATIONS TO RELATIONAL FILES . .	VI-12
VII-1.	PARSE TARGET STRUCTURE FOR "A-POWER-PLANT OF A SUBMARINE WAS-BUILT BY A-COMPANY"	VII-8
VII-2.	MULTIPLE SCRATCH SPACES FOR "A-POWER-PLANT OF A-SUBMARINE WAS-BUILT BY A-COMPANY"	VII-11
VII-3.	VIEWING HIERARCHY ABOVE S2	VII-15
VII-4.	CONTEXT-DEPENDENT PARSE TARGET STRUCTURE FOR "GENERAL.DYNAMICS BUILT THE AMERICAN SUBMARINE"	VII-22
VII-5.	NODE-SPACE PAIRS FOR PHRASES IN "GENERAL.DYNAMICS BUILT THE AMERICAN SUBMARINE"	VII-23
VII-6.	SCRATCH SPACES FOR "GENERAL.DYNAMICS BUILT THE AMERICAN SUBMARINE"	VII-24
VII-7.	THE TWO-NODE INTERPRETATION OF BEAM	VII-31
VII-8.	SCRATCH SPACES WITH EQUI. ARC FOR "THE HENRY.L.STIMSON IS A SHIP"	VII-33
VII-9.	SIMPLIFIED INTERPRETATION OF "THE HENRY.L.STIMSON IS A SHIP"	VII-34

VII-10.	SCR SPACES FOR "DID GENERAL.DYNAMICS BUILD EVERY LAFAYETTE?"	VII-39
VII-11.	ULTIMATE TRANSLATION OF "DID GENERAL.DYNAMICS BUILD EVERY LAFAYETTE?"	VII-42
VII-12.	RESULT OF Q.YN SCOPING PROCEDURE	VII-45
VII-13.	RESULT OF Q.UNIV SCOPING PROCEDURE	VII-46
VII-14.	THE Q FUNCTIONS AND STRENGTHS OF QUANTIFIERS	VII-48
VII-15.	SEMANTIC NET REPRESENTATION OF THE OWNING SITUATION	VII-58
VIII-1.	A SMALL AIR COMPRESSOR	VIII-10
VIII-2.	EXPERIMENTAL SETUP FOR RESTRICTED DIALOGS	VIII-13
VIII-3.	FRAGMENTS OF COOPERATIVE DIALOGS	VIII-16
VIII-4.	DESCRIPTION OF "KNURLED" WITH AND WITHOUT VISION	VIII-18
VIII-5.	USING VISION TO HELP WITH A DESCRIPTION	VIII-18
VIII-6.	DIFFICULTIES IN EXPLAINING AN UNFAMILIAR COMPLEX OBJECT	VIII-19
VIII-7.	PRONOUN USE REFLECTING DIALOG STRUCTURE	VIII-22
VIII-8.	A SEQUENCE OF ELLIPTICAL SENTENCE FRAGMENTS	VIII-24
VIII-9.	A DATA BASE QUERY SUBDIALOG	VIII-25
VIII-10.	A SUBDIALOG CHECKING PREVIOUS MESSAGE	VIII-28
VIII-11.	DIFFERENT USES OF "O.K."	VIII-31
VIII-12.	A MISUNDERSTOOD "O.K."	VIII-33
VIII-13.	A SIMPLE TASK MODEL FOR ILLUSTRATING DIALOG POPS	VIII-38
VIII-14.	SINGULAR/PLURAL DISTINCTIONS	VIII-40
VIII-15.	EFFECT OF SHIFT IN SUBDIALOG ON DEFNPS	VIII-40
VIII-16.	EMBEDDINGS OF REQUESTS AND RESPONSES	VIII-45
VIII-17.	UTTERANCE TYPES IN A SAMPLE DIALOG FRAGMENT	VIII-46
VIII-18.	TWO SIMILAR DIALOG FRAGMENTS FOR COMPARING RESPONSE INFLUENCE	VIII-47

VIII-19.	WORDS OCCURRING IN ALL FOUR DIALOGS	VIII-51
VIII-20.	WORDS OCCURRING IN ALL FIVE DIALOGS, GROUPED BY CATEGORY	VIII-53
VIII-21.	WORDS IN ALL NAIVE APPRENTICE DIALOGS BUT MISSING IN AT LEAST ONE OF THE OTHERS .	VIII-53
VIII-22.	BOLT/NUT CONFUSION	VIII-62
IX-1.	THE KITE STORY	IX-4
IX-2.	A SIMPLE KVISTA WITH TWO FOCUS SPACES	IX-15
IX-3.	QVISTAS FOR "THE WRENCH" AND "THE BOX-END WRENCH" .	IX-15
IX-4.	A KVISTA WITH THE SET OF WRENCHES DIVIDED INTO SEVERAL SUBSETS	IX-17
IX-5.	THE WRENCHES KVISTA WITH FOCUS ADDED	IX-19
IX-6.	PARSE LEVEL SEMANTIC NET REPRESENTATION FOR "AMERICAN SUB"	IX-22
IX-7.	SEMANTIC REPRESENTATION FOR "RED BOX-END WRENCH" .	IX-30
IX-8.	ORIGINAL FOCUS SPACE	IX-31
IX-9.	NEW FOCUS SPACE	IX-32
X-1.	PATH-GROWING ALGORITHM	X-12
X-2.	REPRESENTATIONS FOR "WHAT IS THE SPEED OF THE SUBMARINE?"	X-17
X-3.	RESOLVING "THE CARRIER" AND FIRST LEVEL EXPANSION OF ELLIPSIS TO "THE SPEED OF THE CARRIER" .	X-20
X-4.	FINAL EXPANSION OF ELLIPTICAL UTTERANCE TO "WHAT IS THE SPEED OF THE CARRIER?" . . .	X-21
X-5.	REPRESENTATIONS FOR "DOES BRITAIN OWN THE CARRIER?"	X-23
X-6.	EXPANSION OF THE ELLIPTICAL UTTERANCE, "THE U.S." .	X-24
X-7.	EXPANSION OF THE ELLIPTICAL UTTERANCE, "THE LENGTH"	X-28
XI-1.	SCHEMATIC OF YES/NO QUESTION	XI-4
XI-2.	SCHEMATIC OF WH QUESTION	XI-5

XI-3.	TRANSLATION OF "WHO BUILT THE HENRY.L.STIMSON?" . .	XI-9
XII-1.	KVISTA AND QVISTA FOR THE EXAMPLE QUERY "WHAT SUBMARINES DID GENERAL.DYNAMICS BUILD?" . .	XII-5
XII-2.	AN EXAMPLE QUERY, "WHO BUILT THE HENRY.L.STIMSON?", WHOSE ANSWER IS EXPLICITLY AVAILABLE . . .	XII-7
XII-3.	AN EXAMPLE QUERY, "WHO BUILT THE HENRY.L.STIMSON?", WHOSE ANSWER IS INTERNALLY DERIVABLE . . .	XII-8
XII-4.	AN EXAMPLE QUERY, "WHO OWNS THE HENRY.L.STIMSON?", WHOSE ANSWER IS EXTERNALLY DERIVABLE . . .	XII-10
XII-5.	EXAMPLE KVISTA THEOREM	XII-39
XII-6.	EXAMPLE OF KVISTA EXTRACTION	XII-41
XII-7.	EXTENSION SPACES FOR KVISTA EXTRACTION EXAMPLE . .	XII-42
XII-8.	EXAMPLE OF QVISTA EXTRACTION	XII-49
XII-9.	EXTENSION SPACES FOR QVISTA EXTRACTION EXAMPLE . .	XII-50
XII-10.	RELATING SUMS SITUATION TO FUNCTION PLUS	XII-54
XIII-1.	A SEMANTIC NETWORK FRAGMENT	XIII-4
XIII-2.	THE BASIC GENERATION ALGORITHMS	XIII-6

I INTRODUCTION

Prepared by Ann E. Robinson, Donald E. Walker, William H. Paxton,
and Jane J. Robinson

CONTENTS:

- A. Orientation
- B. An Overview of the Speech Understanding System
 - 1. Components Developed by SDC
 - 2. The Language Definition
 - 3. Syntax
 - 4. Semantics
 - 5. Discourse
 - 6. Deduction
 - 7. Generation
 - 8. Executive
- C. An Example to Illustrate Processing in the System
- D. An Historical Perspective

A. ORIENTATION

For the past five years, SRI has been a part of the Speech Understanding Research Program sponsored by the Advanced Research Projects Agency of the Department of Defense.* The program, begun in 1971 following a thorough assessment of its feasibility by a study group (Newell et al., 1973), launched a multi-disciplinary effort based on state-of-the-art advances in computational linguistics, artificial intelligence, systems programming, and speech science. A set of

* This research has been funded under the following ARPA contracts, all administered through the Army Research Office: DAHC04-72-C-0009, DAHC04-75-C-0006, and DAAG29-75-C-0011.

coordinated, cooperative projects was established to focus further research both in the development of these source knowledge areas and in their effective integration in the context of a complex computer-based system. The goal was to develop one or more systems that would recognize continuous speech uttered in the context of some well specified domain by making extensive use of grammatical, semantic, and contextual constraints. A system emphasizing such linguistic constraints is called a 'speech understanding system' to distinguish it from speech recognition systems, which rely on acoustic information alone.

From the beginning of our participation in the Speech Understanding Research Program, our work at SRI has demonstrated two characteristic features. First, we have approached the problem of natural language processing from the perspectives of artificial intelligence and computational linguistics. Second, we have stressed the importance of having a functioning system guide the progressive elaboration of the various system constituents (Walker, 1973a,b). Following the 1973 midterm review of the ARPA program, we began a joint effort with System Development Corporation (SDC). We were responsible for overall system control and for developing components to handle syntax, semantics, and discourse. SDC was responsible for the acoustic components -- signal processing, acoustic-phonetics, and phonology (see Bernstein, 1975; Ritea, 1975; and Barnett, 1976). This report will concentrate on the SRI contributions both to the operational system that resulted from the joint SRI-SDC effort and to the goal of the ARPA speech understanding program.

Figure I-1 lists the major contributions made by SRI together with the characteristics that distinguish our system from others in the program. These contributions are elaborated further in the following overview of the speech understanding system.

B. AN OVERVIEW OF THE SPEECH UNDERSTANDING SYSTEM

This section contains brief descriptions of the various components of the speech understanding system, including those developed by SDC, and of their coordination by the system executive. Details regarding the components developed by SRI are given in the rest of this report.

The domain for the speech understanding system is information about the ships of the U.S., Soviet, and British fleets. The system data base contains characteristics such as owner, builder, size, and speed for several hundred ships. The user can get information from the system by simple English questions, commands, and dialog sequences using incomplete sentences and pronouns. The internal organization of the system is shown in Figure I-2. The direction of the arrows in the figure indicates the general flow of information as an utterance is interpreted by the system and an appropriate response returned to the speaker.

DEFINITION OF INPUT LANGUAGE

- * Defines the input language by means of linguistically motivated rules that are general and extensible over a variety of domains
- * Provides a means for adjusting ('tuning') the language definition to particular domains without loss of generality
- * Combines syntactic, semantic, and discourse information within the rules that define words and phrases

SEMANTICS

- * Uses partitioned semantic networks
- * Handles higher-order logical predicates, especially quantifiers
- * Provides deduction routines for retrieval and inference that can access supplementary relational data base in responding to a user's query
- * Provides a network substructure that is converted to an English sentence or phrase to answer a user's question

DISCOURSE MODELING

- * Is based on in-depth studies of domain-oriented dialogs
- * Encodes model of dialog context by using semantic partitions
- * Finds meanings of elliptical expressions and referents of definite noun phrases by using dialog context

SYSTEM INTEGRATION

- * Provides for interaction of information from various sources of knowledge -- syntax, semantics, discourse -- as part of the language definition itself
- * Avoids commitment to particular system control strategy, allowing flexible use of various strategies for putting together words and phrases out of incomplete and uncertain fragments

SYSTEM CONTROL

- * Provides special techniques to assign priorities by the use of contextual constraints
- * Allows combinations of top-down, bottom-up, and bidirectional strategies
- * Organizes data structures for testing hypotheses about utterances in a manner that avoids duplication of effort
- * Used in extensive experimental studies to evaluate design alternatives

Figure I-1. SRI CONTRIBUTIONS TO SPEECH UNDERSTANDING

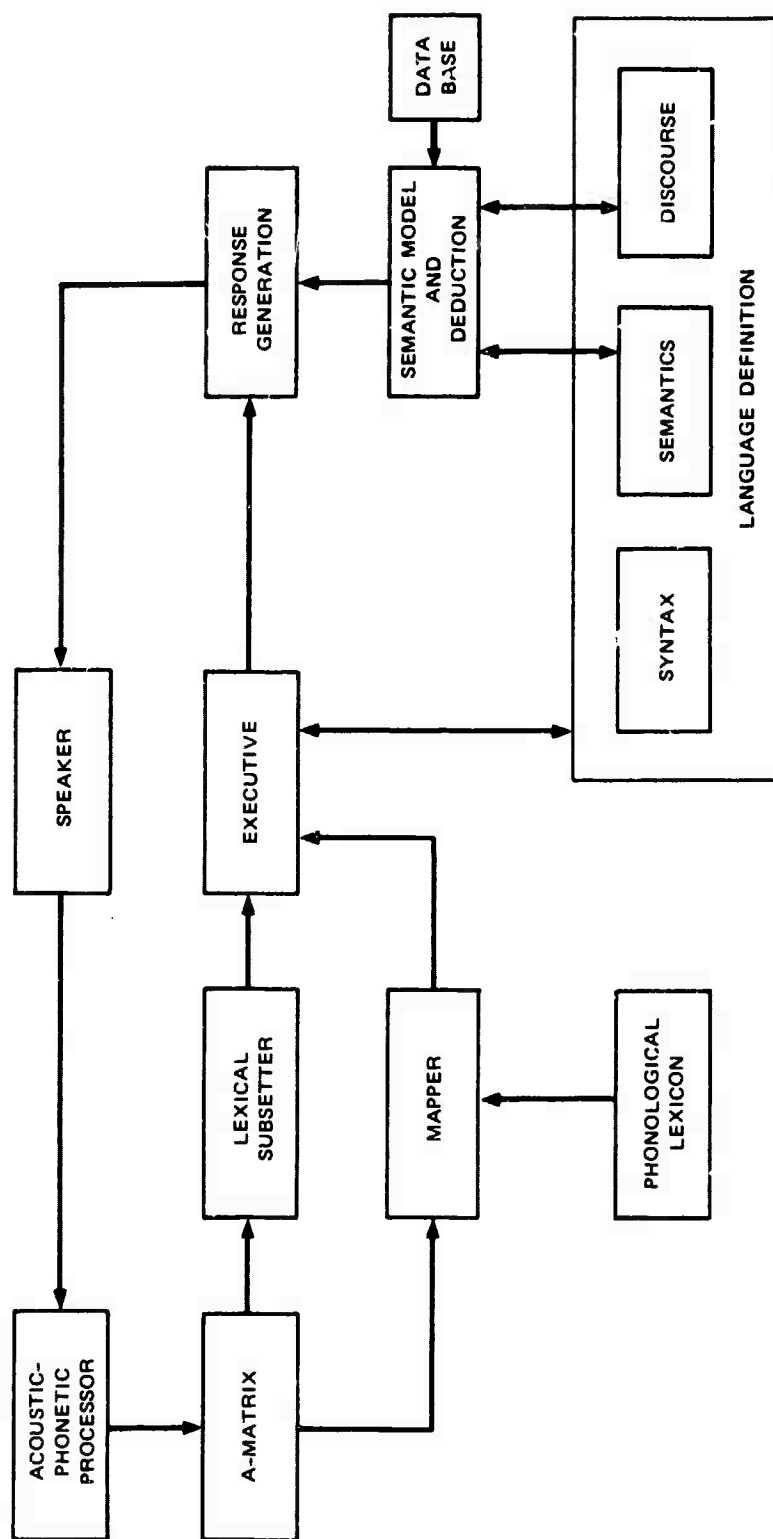


FIGURE I-2 SYSTEM ORGANIZATION

Figure I-2. SYSTEM ORGANIZATION

1. COMPONENTS DEVELOPED BY SDC

The acoustic-phonetic processor, the A-matrix, the mapper, the phonological lexicon, and the lexical subsetter were developed by SDC and are described more fully in their publications referenced above.

The acoustic processor digitizes and records the input from the human speaker at a rate of 20,000 samples per second. RMS-energy values are calculated for each 10 millisecond frame of speech, followed by fundamental frequency extraction, formant frequency analysis, syllable segmentation, phrase segmentation, and other analyses. From these parameters, rough segment labels are derived; subsequent processes use the information available to segment the speech into phoneme-like units, assign feature bits such as nasal or retroflexed, and generate phonemic labels with associated merit scores for each segment. All of the acoustic-phonetic information is stored as an A-matrix for the utterance.

A mapper carries out acoustic tests using the A-matrix data. Given a word predicted by the executive together with a location in the speech input, the mapper compares alternative possible pronunciations of the word with the acoustic data at that point. The location can be specified with a left time, a right time, or both. The mapper assigns a score between 0 and 100 that indicates how well the word matches the input. If the value exceeds a given threshold, the mapper reports the beginning and ending times of the word together with the score.

The lexical subsetter performs an analysis of the A-matrix at a specified location in the utterance and returns a list of words that could begin (or end) at that time. This capability reduces the number of words that otherwise would have to be checked by the mapper.

2. THE LANGUAGE DEFINITION

The input language is a subset of natural, colloquial English that is suitable for carrying on a dialog between a user and the system regarding information in the data base. The definition of this language is based on augmented phrase structure rules. A rule consists of a phrase structure declaration, which specifies the possible constituents of a phrase category, and an augmentation. The augmentation is a procedure containing two principal kinds of statements called 'attributes' and 'factors'. The attribute statements determine the properties of particular instances of a phrase constructed by the rule. An attribute statement may compute values for attributes that relate to syntax, semantics, or discourse. The factor statements compute acceptability ratings for an instance of the phrase. The scores for factors are non-Boolean; that is, they may assume a wide range of values. As a result, a proposed instance of a phrase is not necessarily simply accepted or rejected; it may be rated as more or less acceptable or as more or less 'likely', depending on a combination of factor values. Like attributes, factors may be syntactic, semantic, or discourse related. One of the distinctive features of the language

definition is its integration of the syntactic, semantic, and discourse sources of knowledge through the attribute and factor statements. Another is the provision of non-Boolean factors.

The form of the rules is designed to avoid commitments to particular system control strategies. For example, the rule procedures can be executed with any subset of constituents, so incomplete phrases can be constructed to provide intermediate results, and it is not necessary to acquire constituents in a strictly left-to-right order.

3. SYNTAX

The syntactic knowledge in the system is represented both in the phrase structure part of the language definition rules and in the attribute and factor statements in the procedure part of the rules. Syntax provides computationally inexpensive information about which words or phrases may combine and how well they go together. In testing word or phrase combinations, syntactic information alone often can reject an incorrect phrase without requiring costly semantic and discourse analysis. Factors are used for traditional syntactic tests such as agreement for person or number, but factors also are used to reduce the scores of unlikely phrases. For example, WH-questions that are negative (e.g., "What submarine doesn't the U.S. own?") are unlikely to occur. A factor statement lowers the value for this interpretation but does not eliminate it completely, so that if no better hypothesis can be formed to account for the input utterance, this

interpretation will be accepted. Since the language definition system provides the capability for evaluating phrases in context by means of non-Boolean factors, the grammar can be tuned to particular discourse situations and language users simply by adjusting factors that enhance or diminish the acceptability of particular interpretations. It is not necessary to rewrite the language definition for each new domain.

4. SEMANTICS

The system's knowledge about the domain is embodied in a partitioned semantic network. A semantic network consists of a collection of nodes and arcs where each node represents an object (a physical object, situation, event, set, or the like) and each arc represents a binary relation. The structure of our network differs from that of conventional nets in that nodes and arcs are partitioned into spaces. These spaces, playing in networks a role roughly analogous to that played by parentheses in logical notation, group information into bundles that help to condense and organize the network's knowledge.

Our semantic network also serves as the medium for recording and communicating semantic information among the relevant system components. During the interpretation of an utterance, semantic composition routines, which are called directly from the language definition rules, relate the constituents of a phrase to the network model. These routines build new network structures to reflect the underlying meanings of those phrases that are acceptable and to eliminate those phrases that do not satisfy semantic criteria.

To supplement the knowledge encoded in the network, a relational data base is maintained. It can be accessed directly from the network, which contains a representation of the contents of the data base.

5. DISCOURSE

The discourse component of the speech understanding system relates a given utterance (or a portion of it) to the overall dialog context and to entities and structures in the domain. The current domain of the speech understanding system provides for interacting with information in a data base. In this domain, the discourse context is limited to a linear history of the preceding interactions. For complex task-oriented dialogs, the linear discourse history can be replaced by a structured history mirroring the organization of the task execution.

An important function of the discourse component is to expand elliptical expressions into their full meaning. In our system, a single noun phrase can be accepted as a complete utterance if it can be expanded into a meaningful sentence using information from the previous dialog context. For example, the phrase "The George Washington" is unacceptable in isolation, but following "What is the speed of the Lafayette?", it can be expanded to mean "What is the speed of the George Washington?"

Another important capability is the identification of the referents of definite noun phrases. Partitions in the semantic network are employed to focus the attention of deductive procedures on those items that have been mentioned previously in the dialog. A representation of the referent of a definite noun phrase is kept as the discourse attribute of the phrase. If no referent is found, the phrase is given a low score.

6. DEDUCTION

The deduction component of the system provides an inference mechanism for retrieving information from the semantic network. This component serves a dual purpose. During the interpretation of an utterance, it supplies information needed both by the semantic composition routines and by the discourse procedures. When an interpretation has been found for a question, the deduction component is used to find an answer.

7. GENERATION

The generation component of the speech understanding system contains procedures to produce an English phrase or sentence corresponding to a semantic network substructure. Usually, this substructure is the answer to a question asked by the user. Using a distributed generation grammar, the generator expresses the content of the input nodes and arcs by employing the closest applicable templates

(rules) in the superset hierarchy of those nodes. The answer to a WH question, for example, can be either a noun phrase or a complete sentence, depending on the exact content of the input. The generator can produce a variety of paraphrases of a constituent (e.g., "General Dynamics built the Whale."; "The Whale was manufactured by General Dynamics."). At present, the particular paraphrase is chosen at random.

8. EXECUTIVE

The executive is responsible for coordinating the various components of the system. It uses the language definition and the acoustic components to find an interpretation for the input. When a successful interpretation has been found, the executive invokes the response functions which produce a reply.

The principal data structure used by the executive is called the 'parse net'. It is a network with two types of nodes: phrases and predictions. Phrases in the parse net can be complete, containing all their constituents, or incomplete, with some or all of their constituents missing. A prediction is for a particular category of phrase. Each phrase or prediction is associated with a particular time span in the utterance. As the interpretation of an utterance progresses, new phrases that have been constructed from existing phrases or from words found in the utterance are added to the parse net. At the same time, new predictions are made as more information is obtained. Thus, the parse net grows as the interpretation process advances.

There are two tasks involved in maintaining and evolving this parse net: the 'word' task and the 'predict' task. The role of the word task is to look for a particular word in a particular location in the utterance. If the mapper has not been called previously for that word in that location, the word task calls it. If a word is found successfully in the specified location, the word is used to build a new phrases.

The role of the predict task is to make a prediction for a word or phrase that can help complete an incomplete phrase. Whenever a new constituent is inserted into an incomplete phrase, any adjacent constituents that had been missing can be predicted. Of course, new predictions can include predictions for particular words, leading to new instances of calls on the word task.

Besides creating these tasks, the executive must have a means of determining which one to perform next. Establishing the priority of a task begins with determining the 'score' of the phrase involved. The score is computed from the results of the acoustic mapping of any of the words contained in the phrase, from the factor statements for the phrase, and from the scores of the constituents. After the score is determined, the phrase is given a rating which is an estimate of the best score for a phrase of the root (sentence) category that uses that phrase. This rating is then modified depending on the control strategy being used, and the result is the priority of the task to be performed for that phrase.

C. AN EXAMPLE TO ILLUSTRATE PROCESSING IN THE SYSTEM

We present in this section a partial trace of the successful processing by the system of the question "Who built the Henry L. Stimson?". The utterance is 190 centiseconds (cs) long, extending from 10 to 200 in the A-matrix containing the acoustic-phonetic data for it. To simplify the presentation, interactions with the mapper are not shown.

Lines in the trace beginning with *** indicate nodes that are built into the parse net for complete or partial phrases. If the line ends with ..., the phrase is partial; that is, not all of the constituents are present. Lines beginning with +++ identify nodes in the parse net that specify predictions for the presence of particular categories. Nodes 1 through 10 for phrases and 1 through 49 for predictions do not appear in the trace. These nodes and predictions represent categories that can begin an utterance, and include determiners, nouns, noun phrases, and auxiliaries. They are pre-computed at the time when the language definition is compiled.

In the following presentation, blocks of lines from the trace will be followed by explications of the notation as well as of the processing that takes place.

```

*** 11 *** (N WHALE) 15 25 (9 . 691)=76
*** 12 *** NP UBEG L (17 . 1219)=71 RHS 11 ...
[ WHALE ]

```

CALLING ROUTINE SEMRNP1

RESULT IS NET FRAGMENT (684 (686 683 1))

```

-----
|685| |684|
|1 | |683|
-----
|      |
E<<<<<<<

```

```

*** 13 *** NP 15 25 (17 . 1219)=71 RHS 11
[ WHALE ]

```

The first line of the trace shows that a node (node 11) is built in the parse net for a complete phrase. (A phrase may be a single word.) The (N WHALE) of the first line indicates that the first prediction for which the mapper finds a likely candidate is for a noun (N), and that the candidate was the word "whale". The following pair of numbers, 15 and 25, identify the beginning and ending times, in centiseconds, corresponding to the possible location of the candidate in the utterance. The remaining numbers are parts of the information used for rating the phrase. There are 9 factors whose sum is 691. Therefore, the score is 691 and the quotient is 76. (See Chapter II for an explanation of these terms.)

This complete node is then used by the word task to build a new phrase at node 12 for an NP. In the general case, it is possible to extend a partial phrase by adding constituents on either side. In this case, however, it is not possible to add constituents on the left

because the left anchor coincides within 5 centiseconds of the beginning of the utterance. Thus, the phrase can be extended only to the right, as shown by the UBEG L, which means that the phrase is left anchored at utterance beginning. RHS 11 means that the right-hand-side of the NP production rule to be completed uses the phrase built at node 11 as the first right-hand-side constituent. Following entries for non-terminal nodes, there is a line specifying the terminal words found for it. In this case, there is one word -- "whale" -- for the NP of node 12.

The next step is to build node 13 for a complete NP, using node 11 and the single word "whale". As described in the text, semantic structures are built for a complete phrase as part of applying the rule for that phrase. For this reason, the call on the semantic routine and the results of that call appear before the line for the node. Similarly, discourse calls (e.g., PROCRES for pronoun resolution) appear before the phrases that result from them.

The line CALLING SEMRNP1 indicates that the semantic composition routine SEMRNP1 has been called; the following lines show the resulting semantic net structure built for node 13. The numbers specify the node in the semantic network corresponding to the noun phrase (684) and the space containing the nodes and arcs to which that node is connected (686 683 1), as determined by the composition routine. The boxes contain a node number and, below it, the space the node is on. In this case, the semantic structure of the noun phrase of node 13 is a node in the semantic network numbered 684 on space 683, which is an element of the set 685 on space 1, the set of submarines of the Whale class.

PHRASE 12 RATING IS 840

```
+++ 50 +++ TOKEN.PL 25 L
+++ 51 +++ PREPP 25 L
*** 14 *** PREPP 25 L (2 . 132)=66
+++ 52 +++ PREP 25 L
```

CALLING ROUTINE SEMRNP5

RESULT IS NET FRAGMENT (690 (689 1))
N.LEGAL.PERSONS

```
-----
|690| |10|
|689| |1 |
-----
|      |
E>>>>>>
```

After node 13 is built for a phrase consisting solely of the word "whale", the previous incomplete phrase built at node 12 receives a rating. Subsequently, the executive makes predictions for a plural (TOKEN.PL) and for a prepositional phrase (PREPP) to the right of the noun phrase in node 12. These predictions, 50 and 51, begin at time 25 in the input. A parse net node (14) for an incomplete phrase (with no constituents) is built for the PREPP prediction. It causes another prediction to occur for a preposition (PREP) at the left of the phrase, corresponding to the phrase structure part of the preposition rule, which is PREPP = PREP NP. The preposition "by" is found in the input by the mapper (this is not shown in the trace), and so a new node is built, which is partially completed by the addition of "by". However, when this phrase is evaluated in the context of the NP from node 12 which caused this prediction, the attribute and factor statements determine that "by" is not a valid preposition to follow the noun "whale", and so the phrase built in node 12 is rejected.

```

*** 15 *** (NP WHO) 15 25 (3 . 213)=71
*** 16 *** S1 UBEG UEND (11 . 755)=68 RHS 15 ...
[ WHO ]
PHRASE 16 RATING IS 823

```

```

CALLING ROUTINE PRONRES
RESULT IS NET FRAGMENT (15 (1))
N.THE.US
      N.COUNTRIES

```

```

-----
|15 | |12 |
|1  | |1  |
-----

```

```

|
E>>>>>>

```

```

*** 17 *** (NP WE) 15 25 (3 . 195)=65
+++ 53 +++ BE 25 L
+++ 54 +++ DO 25 L
+++ 55 +++ VP 25 L
*** 18 *** VP 25 L (6 . 396)=66
+++ 56 +++ V 25 L
(DO.THEY (8 . 511) 30 (1 . 1) 75)
*** 19 *** (DO DO) 30 (1 . 1) (8 . 511)=63
*** 20 *** S1 UBEG UEND (18 . 1200)=66 RHS 15 19 ...
[ WHO DO ]
PHRASE 20 RATING IS 800
+++ 57 +++ NP (1 . 1) L
CALLING ROUTINE PRONRES
RESULT IS NIL: REJECT!

```

The next word found ("who") satisfies an initial prediction for a noun phrase. It extends from 15 to 25. A semantic structure is built for it (node 15) which indicates that it refers to some unspecified member of the set of legal persons. Node 16 is built for an incomplete S (sentence) phrase with the NP "who" at the beginning.

The next word found is "we". A discourse routine (PRONRES) is called to find a referent for it, and "the U.S.", which is an element of the set of countries, is found. Thus, another noun phrase is found for the initial portion of the utterance.

The next predictions made, numbers 53-55, are for the various constituents that can complete phrases starting with an NP. They are provided by the rule S1 in the grammar, which allows the following patterns:

```
S = NP DO NP VP
S = NP VP
S = NP BE {VP | NP | "THERE"}
```

The VP prediction leads to the construction of a node for a verb phrase and to the further prediction of a V. The multi-word "do.they" is found satisfying prediction 54, and a node is constructed using "do". This is distributed to the S rule, and a new S node (20) is constructed with the words "who do" in it. The discourse routines reject the attempt to use "they" to construct an NP.

```
*** 22 *** (V BUILT) 30 70 (9 . 572)=63
*** 23 *** VP 25 L (14 . 902)=64 RHS 22 ...
[ BUILT ]
PHRASE 23 RATING IS 0
*** 25 *** (V BUILT) 30 70 (9 . 572)=63
*** 26 *** VP 25 L (14 . 902)=64 RHS 25 ...
[ BUILT ]
*** 27 *** VP 30 70 (14 . 902)=64 RHS 25
[ BUILT ]
PHRASE 26 RATING IS 795
+++ 58 +++ TOKEN.PPL 70 L
+++ 59 +++ TOKEN.PAST 70 L
+++ 60 +++ TOKEN.SG 70 L
+++ 61 +++ PREPP 70 L
*** 28 *** PREPP 70 L (2 . 132)=66
+++ 62 +++ PREP 70 L
+++ 63 +++ NP 70 L
*** 29 *** NP 70 L (9 . 594)=66
+++ 64 +++ TOKEN.A 70 L
+++ 65 +++ WHDET 70 L
+++ 66 +++ DET 70 L
+++ 67 +++ TOKEN.HOW.MANY 70 L
+++ 68 +++ N 70 L
+++ 69 +++ CLASSIFIER 70 L
+++ 70 +++ NUMBER 70 L
*** 30 *** NUMBER 70 L (4 . 264)=66
```

+++ 71 +++ CENTI 70 L
 *** 31 *** CENTI 70 L (4 . 264)=66
 +++ 72 +++ SMALLNUM 70 L
 *** 32 *** SMALLNUM 70 L (5 . 330)=66
 +++ 73 +++ DIGIT 70 L
 +++ 74 +++ TEEN 70 L
 +++ 75 +++ TOKEN.HUNDRED 70 L
 +++ 76 +++ TOKEN.THOUSAND 70 L

CALLING ROUTINE PRONRES
 RESULT IS NIL: REJECT!

*** 34 *** (DET THE) 70 75 (1 . 61)=61
 *** 35 *** NP 70 L (9 . 589)=65 RHS 34 ...
 [THE]

PHRASE 35 RATING IS 792

+++ 77 +++ NUMBER 75 L
 *** 37 *** NUMBER 75 L (4 . 264)=66
 +++ 78 +++ CENTI 75 L
 *** 38 *** CENTI 75 L (4 . 264)=66
 +++ 79 +++ SMALLNUM 75 L
 *** 39 *** SMALLNUM 75 L (5 . 330)=66
 +++ 80 +++ DIGIT 75 L
 +++ 81 +++ TEEN 75 L
 +++ 82 +++ TOKEN.HUNDRED 75 L
 +++ 83 +++ TOKEN.THOUSAND 75 L
 +++ 84 +++ N 75 L
 +++ 85 +++ CLASSIFIER 75 L
 *** 40 *** (N HENRY.L.STIMSON) 80 195 (11 . 836)=76
 *** 41 *** NP 70 L (19 . 1359)=71 RHS 34 40 ...
 [THE HENRY.L.STIMSON]

CALLING ROUTINE SEMRNP1
 RESULT IS NET FRAGMENT (301 (694 693 1))
 HENRY.L.STIMSON
 N.CLASS.LAFAYETTE

301	109
1	1

E>>>>>>

*** 42 *** NP 70 195 (19 . 1359)=71 RHS 34 40
 [THE HENRY.L.STIMSON]

The next successful mapping of a predicted word is "built". It occurs twice because it has two senses, past and passive. The passive form is quickly rejected by the case factors but the past is retained. Two VP phrases are constructed, nodes 26 and 27. Node 27 can include only the verb since its boundaries coincide with that of the word, but 26 can have another constituent. Predictions 58 through 76 are predictions for constituents following the V in phrase 26. Note that nodes 28-32 also are constructed in the process of making these predictions in turn, lead to predictions.

After these predictions are made, the mapper is called to look for possible words beginning at 70, and the mapper finds the word "it" (not shown in the trace). The discourse routines reject "it", because there is no referent for it in the current context.

The word "the" is then found; phrases 34 and 35 are constructed; and predictions 77 through 85 are made for possible next constituents in the NP phrase. "Henry L. Stimson" is found after "the". The semantic structure built for the phrase "the Henry L. Stimson" contains a pointer to the node for the individual ship the "Henry L. Stimson" and identifies it as an element of the "Lafayette" class.

```
CALLING ROUTINE SEMRVP1
RESULT IS NET FRAGMENT (697 (698 696 694 693 1))
      HENRY.L.STIMSON
            S.BUILD
```

```
-----
|697| |301| |187|
|696| |1 | |1 |
-----
|
|OBJ>>>>
|
```


PHRASE 41 RATING IS 824
+++ 86 +++ TOKEN.PL 195 L
+++ 87 +++ PREPP 195 L
*** 45 *** PREPP 195 L (2 . 132)=66
+++ 88 +++ PREP 195 L

(GENERAL.DYNAMICS)
44
WHO BUILT THE HENRY.L.STIMSON
824
S1 44
NF WHO 15
VP 43
V BUILT 25
NP 42
DET THE 34
N HENRY.L.STIMSON 40

Node 44 is the completed phrase. The system continues processing to determine if there are any other competing interpretations that could be better than this one. There are none, so the interpretation and its answer are accepted as correct.

At the end of the example is the parse tree. It shows that the structure found is an S, which consists of an NP and a VP. The VP consists of a V and another NP. This second NP consists of a DET and an N.

D. AN HISTORICAL PERSPECTIVE

Reflecting our concern with the importance of implementing a complete system as early as possible, our first system adapted an existing language understanding program designed for text input (Walker, 1973a,b). However, although our initial results were positive, it became clear that for processing spoken utterances many more alternative possible interpretations of their structure have to be considered. The uncertainties associated with segmenting and labeling the acoustic input in continuous speech contrast markedly with the easy identifiability of words in texts. To provide the required flexibility, our second system featured a new parsing strategy that attempted to explore the most likely parse paths first (Paxton and Robinson, 1973; Paxton, 1975). We were able to reduce the size of the search space in this way, thus avoiding the inefficiencies of both depth-first and breadth-first parsing. We also began the development of our work on performance grammars (Robinson, 1975a) and on the systematic analysis of task-oriented dialogs (Deutsch, 1974a). A case subsystem was introduced to provide more sophisticated semantic processing, and functions were developed to resolve simple anaphoric reference and to correlate information from a primitive world model. Using programs for speech analysis and word verification developed by the SRI Sensory Science Research Center (Becker and Poza, 1975), we were able to process 71 utterances with an accuracy greater than 60% (Walker, 1974, 1975).

Following the completion of this system and the mid-period review of the ARPA Program, we began our joint effort with SDC. (Walker et al., 1975; Paxton and Robinson, 1975; Robinson, 1975b; Hendrix, 1975c; Deutsch, 1975; Slocum, 1975; Paxton, 1976a, 1976b). The components of the speech understanding system that were developed by SRI were programmed initially in INTERLISP-10 (Teitelman, 1975) on a PDP-10 TENEX system. In the system implementation at SDC, the acoustic processing was performed on a PDP-11, and the rest of the system ran on an IBM 370/145 under the VM operating system. We were able to use INTERLISP/370 (Uppsala University, 1975)* for the SRI components, which simplified the transfer of programs. The mapper was programmed in CAP (Barnett and Pintar, 1974), an assembly language developed by SDC.

The results of this cooperative effort culminated in the system that is described in this Final Report. However, immediately after we had brought up an operational system, the SDC computer facility was removed and further refinement of the system as a whole no longer was possible. During the last week before the removal of the SDC computer, we were able to get data on the performance of the acoustic components of the system. Subsequently, we have conducted extensive tests of the system framework, simulating the unavailable acoustic components. These results also are presented here.

* We are grateful to Jaak Urmi and the Uppsala University Computer Center for their help in installing INTERLISP-370 on the SDC Computer.

In the succession of speech understanding systems described above, we dealt with several domains that differed in size and complexity. In the systems developed wholly at SRI, we began with the blocks world, then worked on the repair of plumbing fixtures. With SDC, we were to have dealt both with the maintenance of electromechanical equipment (taking advantage of a companion project at SRI that was developing a computer-based consultant*) and with operations on a file containing information about the attributes of ships in different naval fleets (which SDC had worked on earlier). We developed strategies for the maintenance problem, but a general reduction in ARPA funding limited our resources, and further activities in that area were postponed. Our current system uses the navy ships domain, and most of the work described in this report will reflect that context.

The work on speech understanding at SRI has produced a system design concept and a set of natural language processing components that are well-suited for research on natural language understanding generally. Chapters II and III present detailed descriptions of the Definition System and the Executive System that provide overall integration and control. Chapter IV discusses the results of the experiments that we conducted to test alternative system control strategies. Chapters V, VI, and VII describe the representation of semantic knowledge, present a model of the domain, and show how semantic processing is used in the interpretation of an utterance.

* See Nilsson et al., 1975; Hart, 1975.

Chapters VIII, IX, and X deal with discourse and include discussions of dialog collection and analysis, the resolution of definite noun phrases, and ellipsis. Chapters XI, XII, and XIII indicate how the system responds to the interpreted utterance, how deduction is used both to find an answer and in the interpretation process, and how the system generates replies in English to a user. Following the references is a complete list of publications and reports produced under the various ARPA contracts that have supported our research.

II THE DEFINITION SYSTEM

Prepared by William H. Paxton

CONTENTS:

- A. Introduction
- B. The Metalanguage
 - 1. Composition Rules
 - 2. The Lexicon
 - 3. Global Declarations
 - 4. Annotated Formal Syntax
- C. A Version of the SRI Language Definition
 - 1. Global Declarations
 - 2. Lexicon
 - 3. Composition Rules
- D. The Definition Compiler
 - 1. Category Records and the Lexicon
 - 2. Rule Records, Structure Graphs, and Procedures
 - 3. Details of Rule Compilation Algorithms
 - 4. Lookahead Information
- E. Discussion

A. INTRODUCTION

This chapter contains a detailed discussion of the Definition System. The Definition System consists of a metalanguage for writing definitions of the input language for the speech understanding system and a compiler to convert such definitions into a form for use by the Executive System.* In this chapter, the metalanguage is described, and

* We make the usual distinction between the metalanguage and the object language: the object language is the language being defined (in our case, it is the system's input language); the metalanguage is the language used to state the definition. The 'language definition' is written in the metalanguage and specifies the object language.

its use is illustrated by a sketch of the SRI language definition. The final part of the chapter, Section D, contains a discussion of the Definition Compiler, focusing on the process of rule translation and describing the internal representation of the structural and procedural information. The use of the translated language definition in understanding utterances is described in Chapter III, The Executive System.

B. THE METALANGUAGE

The metalanguage is designed for specifying the definition of the input language for the speech-understanding system. Such a language definition consists of a lexicon containing the vocabulary, a set of composition rules for combining words into phrases and smaller phrases into larger ones, and some global declarations giving information needed by the Definition Compiler and the Executive. The lexicon is separated into categories, such as noun and verb, and the words in each category are assigned values for various attributes such as grammatical features and semantic representation. The composition rules are phrase structure rules augmented by a procedure which is executed whenever the rule constructs a phrase. Information provided by the procedure includes both attributes of the phrase based on the attributes of its constituents, and factors for use in judging the acceptability and likelihood of the phrase. The global declarations in a language definition give information such as lists of attributes for the different categories.

1. COMPOSITION RULES

A speech-understanding system uses several kinds of knowledge, each playing a particular role during the processing of an utterance. For example, our system employs knowledge about acoustics, syntax, semantics, and discourse. The composition rules in the language definition are the principal means by which these knowledge sources are integrated. In addition to defining the possible constituent structure for phrases, each rule has a procedure for calculating both attributes of phrases and factors for use in judging phrases. Phrases with their attributes and factors are the basic units for the integration of knowledge sources in our system. Because the rule procedures may call upon any or all of the sources of knowledge, the attributes and factors of a phrase can, and generally do, reflect decisions by each major component from acoustics to discourse. The following paragraphs describe the structure of composition rules; Section C.3 of this chapter contains more details about a complete set of rules for a small language definition.

Part of a composition rule is shown in Figure II-1. The rule starts with the keyword `RULE.DEF` followed by the rule name (`S1`), the structure declaration, and the procedure. In the structure declaration, vertical bars separate alternatives, braces are used to delimit a set of alternatives, parentheses delimit optional items, and angle brackets mark an optional set of alternatives. Category names can be terminated with a number to provide unique names for different

```

RULE.DEF S1  S = NP1 <(DO NP2) VP1 | BE {VP2 | NP3 | "THERE"}>;

BEGIN
.

MOOD = IF DEIX(NP1) EQ "WH THEN "WH
      ELSE IF DEIX(NP1) NQ "UNDEFINED THEN "DEC
      ELSE "UNDEFINED;
IF MOOD EQ "WH THEN F.MOOD = GOOD;
IF OMITALL(VP1,BE) AND SUBCAT(NP1) EQ "PRO AND MOOD EQ "DEC
  THEN F.REJECT(F.PROSENT);
.

END;

```

Figure II-1. PART OF A COMPOSITION RULE

occurrences in the rule of the same category (e.g., NP1, NP2, and NP3, are all noun phrases or NPs). A quote mark indicates that the next word is to be taken literally rather than being interpreted as a category name. Thus, the phrase structure declaration in Figure II-1 states that a phrase of category S can be composed of a noun phrase, NP1, optionally followed by either a predicate with a verb phrase, VP1, or a predicate with a BE verb (such as "is" or "are"). The constituent VP1 can optionally be preceded by a DO verb (such as "did" or "does") and a noun phrase, NP2. The BE verb must be followed by either a verb phrase, VP2, a noun phrase, NP3, or the word "there".

The portion of Figure II-1 starting with the word "BEGIN" contains an excerpt from the procedure for the rule. The first statement assigns a value to the MOOD attribute. The expression DEIX(NP1) refers to the attribute named DEIX of the constituent NP1. If

the value of the DEIX attribute of NP1 is WH, the MOOD attribute of the sentence is set to WH (indicating a question like "What ..." or "Who ..."). The MOOD is set to DEC (indicating a declarative sentence) if DEIX of NP1 is not WH and is not UNDEFINED. (The default value of attributes is the special symbol UNDEFINED.) However, if DEIX of NP1 is UNDEFINED, MOOD is also set to UNDEFINED. The next statement sets the MOOD factor, F.MOOD, to GOOD if the MOOD attribute of the sentence is WH. This is a nonBoolean factor indicating a high expectation for WH questions. The last statement in the figure is a restriction blocking elliptical sentences formed of a single nonWH pronoun such as "we". In other words, if both kinds of predicates are omitted (OMITALL(VP1,BE)), if NP1 is a pronoun (SUBCAT(NP1) EQ "PRO"), and if the sentence is declarative (MOOD EQ "DEC), then the phrase is blocked (F.REJECT(F.PROSENT)). The full procedure for the rule contains several pages of such attribute and factor statements.

In this and other rules, there are attributes that specify acoustic properties related to the input signal, syntactic properties such as mood and number (singular or plural), semantic properties such as the semantic network representation of the meaning of the phrase, and discourse properties for anaphora and ellipsis. The values of constituent attributes are used in computing the attributes of larger phrases, and the attributes of complete interpretations are used in generating responses.

The factors also use acoustic, syntactic, semantic, and discourse information. Acoustic factors reflect how well the words match the actual input, syntactic factors deal with tests such as number agreement between various constituents, semantic factors ensure that the meaning of the phrase is reasonable, and discourse factors indicate whether an elliptical or anaphoric phrase makes sense in the given dialog context. The values of factors are included in a composite score for the phrase. The scores of constituents are combined with the factor scores to produce the scores of larger phrases, and the scores of complete interpretations are used in setting Executive priorities.

Attributes and factors either have constant values or have values that depend on attributes of constituents and global information (such as a model of the discourse or the results of preliminary, low-level acoustic processing). By design, the attributes and factors of a phrase are not allowed to depend on the context formed by other phrases that can combine with it to produce larger structures. This restriction makes it possible to share phrases among different contexts and reduces duplication of effort in the Executive.

Another restriction on the rule attribute and factor definitions is that they must cover cases in which the value of a referenced attribute has the special value UNDEFINED. The primary reason for UNDEFINED attributes is the desire to allow Executive control strategies that depend on information regarding incomplete phrases -- phrases missing one or more constituents. With the attribute and factor

definitions required to handle UNDEFINED attributes of missing constituents, the Executive can execute the rule procedure with a partial set of constituents, and the results will be indicative of possible completions of the phrase. The use of this ability in setting priorities for the Executive is a topic of Chapter III, Section D.5.

There is an emphasis on factors in the language definition because of the need to block bad phrases that might be incorrectly accepted by acoustic tests. A system with text input can usually tolerate a language definition that accepts a wide variety of strange combinations of words as long as the looseness of the definition does not produce apparent ambiguities in actual user inputs. In other words, the text system can focus on what the user might say and generally ignore what he/she will not say. A language definition for a speech-understanding system should be general enough to allow the speaker to communicate naturally, but it must also block unacceptable phrases that might be incorrectly 'heard' due to errors in acoustic tests even though no speaker would actually say such phrases.

2. THE LEXICON

Like the composition rules described above, the lexicon combines declarative and procedural information. However, while the rules are predominantly procedural, the lexicon mainly contains static declarations of words and their attributes. The structure of the lexicon is illustrated by considering the information for the word "length".

```

WORDS.DEF N

WORDFN LAMBDA(C) NULLDEFAULTATTRS(C,"(DETREQ MEAS RELN UNIT INDFLG));
SUBCATEGORY RELN.MEASURES
  ATTRIBUTES MEAS=T, NBR=SG, ....;
WORDS
  LENGTH
    PDGM=(S.HAVE.LENGTH PG.INVH),
    SUPSET=N.LENGTHS;
  SIZE
  .
  .
ENDWORDS;
END;

```

Figure II-2. SAMPLE LEXICAL ENTRY

Figure II-2 contains an extract from the SRI lexicon containing information related to "length". "Length" is in the subcategory RELN.MEASURES (relational measures) of the lexical category N (nouns). Some other RELN.MEASURES are "size" and "speed". The noun subcategories correspond to semantic classes that are important in the task domain of the speech system. Attributes declared for a subcategory are shared by all of its members. Thus, because it is a RELN.MEASURE, "length" is automatically given several attributes including one that marks it as a measure term (MEAS=T) and another that marks it as singular (NBR=SG). Default values for some other attributes are shared by all the N subcategories. For instance, because "length" is not marked otherwise, the WORDFN redundancy function for N sets attributes DETREQ, UNIT, and INDFLG, to the value NIL to indicate, respectively,

that "length" does not require a determiner, that it is not a unit of measurement (such as "feet"), and that it does not refer to an individual (such as "England"). Most of the attributes of "length" are set according to category and subcategory redundancies. The only attributes explicitly given for the particular word "length" are PDGM and SUPSET which relate to its meaning, the information that distinguishes "length" from other RELN.MEASURES. (Phonological information that would also distinguish "length" is stored separately since it is only used in acoustic processing.)

In addition to word and attribute declarations, lexical categories have an associated procedure that is invoked whenever a word from the category is found in an utterance. For example, the lexical NP procedure, for words like "it" and "who", calls semantic routines to build nodes in a semantic network and also calls discourse routines to find possible referents.

3. GLOBAL DECLARATIONS

The global declarations at the start of a language definition provide information needed by the Definition Compiler and the Executive. This information includes a list of categories to be used in the definition and lists of attributes for the categories. The global declarations can also contain redundancy functions for rewriting category and rule definitions. These functions were included in the design because they provide ways to simplify the definition in much the

same way that macros can simplify a program; however, they have not been used in the current system.

4. ANNOTATED FORMAL SYNTAX

As described above, a language definition contains a lexicon, composition rules, and global declarations. The lexicon groups words into categories and subcategories. The composition rules have a phrase structure declaration augmented by a procedure specifying attributes and factors. The global declarations provide information needed by the Compiler and the Executive. The remainder of this section is devoted to a more formal statement of the structure of a language definition.

An annotated formal syntax of the metalanguage is given below using phrase structure rules with the notation described previously. Vertical bars separate alternatives, braces delimit a set of alternatives, parentheses enclose an optional set of items, angle brackets bound an optional set of alternatives, and a single preceding quote mark indicates a literal. Any item whose name ends with the string "name" is an identifier, and items with names ending with the string "names" refer to a series of one or more identifiers separated by commas. In this formalism, the first part of a language definition is shown in Figure II-3.

The global declarations include a list of categories, the name of the root category (typically the sentence category, S), specification

```

language.def = "LANGUAGE.DEF decls "END "; rules.and.categories
decls = decl "; (decls)

decl = "CATEGORIES categorynames | "ROOT "CATEGORY categoryname |
      decl.function functionspec |
      "ATTRIBUTES attr.decls "ENDATTRS

decl.function = "RESPONSEFN | "SCOREFN | "WORDFN |
               "CATEGORYFN | "RULEFN

functionspect = functionname |
               "LAMBDA "( (variablenames) ") expression

attr.decls = attr.decl "; (attr.decls)

attr.decl = "{categorynames | "ALL ("EXCEPT categorynames)}
           {"HAVE | "HAS} attributenames

rules.and.categories = {lexical.category | composition.rule}
                      (rules.and.categories)

```

Figure II-3. DECLARATIONS

of various functions, and declarations of attributes. The RESPONSEFN function is called by the Executive whenever a root category phrase is constructed or when some resource limit is reached. The SCOREFN function is responsible for combining individual factor values into a composite rating for the phrase. (The particular procedures used for RESPONSEFN and SCOREFN in the speech understanding system are described in Chapter III, Sections C.4 and D.5. .) The CATEGORYFN, RULEFN, and WORDFN, are functions that make changes in the definitions for lexical categories, composition rules, and words, respectively, before the definitions are compiled. The expression appearing in the function specification is an arbitrary LISP expression written in an infix

notation developed at SDC (see Barnett, 1973). The attribute declarations give lists of the various categories and their attributes for use by the Compiler.

The syntax for the lexicon is shown in Figure II-4. The optional expression in the lexical category specifies the category procedure. Following it can come a WORDFN function to modify the word definitions in the category before they are compiled. A typical use of a WORDFN is to supply default values for attributes. A category definition can contain either a set of words or a series of subcategories. Each word can have an arbitrary number of attribute-value pairs. Each subcategory can have a set of attribute-value pairs in addition to its set of words. These attribute-value pairs provide defaults for the words in the subcategory. For example, if attribute A is listed with value B in the subcategory attributes, all words in the subcategory that do not explicitly assign a value to A get B as a default assignment. The attribute values in the lexicon are LISP data items such as atoms, numbers, or lists.

The syntax for composition rules is shown in Figure II-5. The rule structure declarations use the same notation for phrase structure as is employed in this section. The rule subfunctions and the rule expression form the procedural part of the rule. They are written in a dialect of LISP (see Barnett, 1973) with extensions for testing constituent structure and computing attribute and factor values. The Definition Compiler recognizes references to attributes by means of the

```

lexical.category = "WORDS.DEF categoryname (expression ");
                    lexcatparts "END ";

lexcatparts = {"WORDFN functionspec |
              "WORDS catwords "ENDWORDS |
              "SUBCATEGORY subcatname
                ("ATTRIBUTES catwordattrs ");
                catwords "ENDWORDS}
              "; (lexcatparts)

catwords = lexentryname (catwordattrs) "; (catwords)

catwordattrs = attributename "= attributevalue (" , catwordattrs)

```

Figure II-4. LEXICON

```

composition.rule = "RULE.DEF rulename structure
                   (subfunctions) expression "END ";

structure = categoryname "= rhsalts ";

rhsalts = rhsseries ("| rhsalts)

rhsseries = rhsitem (rhsseries)

rhsitem = "( rhsseries " ) | "{ rhsalts " } | "< rhsalts "> |
          "" literalname | categoryname

subfunctions = "RULE.SUEFN functionname
               "( (variablenames) " ) expression ";
               (subfunctions)

```

Figure II-5. RULES

global declarations of their names. Factor names are identified by an "F." prefix. A rule application can be blocked by the statement "F.REJECT(factorname)". This statement causes immediate termination of the rule. Both attributes and factors can be used in expressions and

can be assigned values. Attributes of constituents can be accessed by an expression of the form "attributename(constituentname)". Attributes are often set to the same value as an attribute of the same name in some constituent, so a special statement is provided for this operation:

`^ATTRS attributenames FROM constituentname.`

This statement produces for each attribute in the list an assignment statement of the form

`attributename=attributename(constituentname).`

The main forms for testing constituent structure are "HAVE constituentname" and "OMIT constituentname". HAVE implies that the constituent position is filled with a phrase. OMIT implies that the constituent position is not going to be filled because some other alternative has been selected. The rule procedures are sometimes invoked by the Executive with only a partial set of constituents, and it is possible in such cases for both HAVE and OMIT to be false for a missing constituent. Once the phrase is complete, however, either HAVE or OMIT, and not both, will be true for each constituent. For tests with HAVE and OMIT that refer to more than one constituent, logical connectives AND, OR, and NOT are available, or one of the following special operators can be used: HAVEALL, HAVEANY, OMITALL, and OMITANY. These operators take a list of constituent names as arguments and have the obvious meanings.

C. A VERSION OF THE SRI LANGUAGE DEFINITION

A version of the SRI language definition will serve as an illustration of the use of the Definition System. The definition described below is of moderate complexity: it is less complex than those used in some current natural language text systems, but more complex than the languages of most previous speech systems. It was derived from a larger definition and used in the series of experiments described in Chapter IV.*

The domain of discourse for the language is a data base of information about ships of the U.S., Soviet, and British fleets. The particular domain of discourse determines a large portion of the vocabulary, and, hence, the lexicon. A change in the domain would require corresponding changes in the vocabulary and lexicon. The composition rules, however, are quite general and the effect on them of a change in discourse domain would be relatively small. Some attributes and factors in the rules have been 'tuned' to the particular domain (see Robinson, 1975), but most of them deal with general features of English.

There is information in the data base about several hundred ships and a large number of ship classes and categories. For each ship, the data base contains characteristics such as name, type, owner, builder,

* The larger definition was developed by Jane Robinson and Ann Robinson, with assistance from Gary Hendrix, Joyce Friedman, and myself. As designer of the Definition System, I influenced the general structure of the definition but did not work out the details. After the definition was relatively complete, I extracted a subset, made some revisions, and used the result in a series of experiments.

length, beam, draft, displacement, speed, complement, and power. The language definition is designed to allow a user to get information from the data base by questions, commands, and dialog sequences using incomplete sentences and pronouns. The 60 test sentences used in the experiments are listed at the end of Chapter IV. These sentences indicate the scope of the language in an informal way. The following paragraphs give a more precise description.

1. GLOBAL DECLARATIONS

Figure II-6 contains an abbreviated version of the global declarations. There are 18 categories with S as the root category. There are 41 attributes, which can be divided into four sets: 12 attributes for syntax, 13 for case semantics, 9 for semantic translation, and 7 for discourse. The category with the most attributes is NP with 24. On the average, each category has about eight attributes. The RULEFN and SCOREFN are not given explicitly since the system defaults are used. There are also no redundancy functions defined for category, word, or rule definitions.

2. LEXICON

The lexicon is divided into twelve categories. There are three categories of verbs, BE, DO, and V, illustrated by "is", "does", and "own", respectively. All have attributes for number (singular or plural) and tense (present or past), and verbs in category V also have

```

LANGUAGE.DEFINITION
  CATEGORIES S,NP,VP,CLASSIFIER,PREPP,PREP,N,BE,DO,V,
              NUMBER,CENTI,SMALLNUM,TEEN,DIGIT,DET,WHDET,ADJ;
  ROOT CATEGORY S;
  ATTRIBUTES
    S HAS REPLY;
    V,VP,BE,DO HAVE TENSE;
    NP,ADJ,VP,WHDET,DET,PREPP,NP HAVE SUPSET,SUPCASE;
    .
    .
  ENDATTRS;
END;

```

Figure II-6. GLOBAL DECLARATIONS

attributes for voice (active or passive) and case semantics that resemble the case grammar of Fillmore (1968) as adapted to computer use by Celce Murcia (1976) and others. There are two categories of numbers: DIGIT and TEEN. The TEENS ("ten", "eleven", and "twelve") are separate because they do not combine in the same manner as DIGITs to form larger numbers (for example, 31 cannot be said as "twenty eleven"). Both DIGITs and TEENS have attributes giving their numeric value and their grammatical attributes. Determiners are also split into separate groups to simplify the rules: declarative determiners like "the" are in the category DET; question (WH) determiners like "what" are in the category WHDET; and the indefinite "a" is included as a literal in the noun phrase rule. The categories for adjective (ADJ) and prepositions (PREP) also appear in the lexicon, but are represented by only two words each: "of" and "by" for PREP, "fast" and "long" for ADJ.

The final three lexical categories (N, CLASSIFIER, and NP) are each divided into subcategories. There are two subcategories of NPs: countries (such as "Russia") and pronouns (like "it"). In both cases, the words have attributes similar to those for a noun phrase constructed by the NP composition rule. These attributes include number (singular or plural), case (nominative or accusative), and semantic interpretation. The lexical NP procedure also calls the discourse routines to find possible referents for the pronouns; it blocks use of the pronoun if no referent is found.

Classifiers are prenominal modifiers. There are three subcategories of classifiers in the lexicon: countries (as in "British ships"), type designations (as in "nuclear submarine"), and predicates (as in "patrol sub"). All the classifiers have attributes indicating the kinds of nouns they can modify and their semantic translation.

By far the largest lexical category is N, nouns. There are 10 subcategories of N: units of measure (such as "ton"), parts of ships (such as "reactor"), classes of ships (such as "Nautilus"), individual ships ("Sealion"), companies ("General Dynamics"), countries ("England"), relational measures ("length"), two kinds of ship types ("CGN", "submarine"), and a subcategory of miscellaneous nouns. Members of category N have grammatical attributes like number, semantic attributes such as pointers into a semantic network, and attributes used for both syntax and semantics such as information regarding whether the N is a measure, a unit, or a relation.

3. COMPOSITION RULES

In addition to the twelve lexical categories, the language definition includes ten composition rules. First to be discussed are the three number rules whose phrase structure declarations are given in Figure II-7. SMALLNUMs can be a DIGIT ("one"), a TEEN ("eleven"), a DIGIT followed by the suffix TEEN ("fourteen"), a DIGIT followed by the suffix TY ("seventy"), or a DIGIT TY DIGIT sequence ("sixty four"). The two occurrences of TEEN and DIGIT in the phrase structure are disambiguated by use of numeric suffixes, "1" and "2". Thus, the rule procedure refers to the first (leftmost) DIGIT as DIGIT1, and the second, as DIGIT2. The SMALLNUM procedure checks attributes on the digits since some cannot be followed by TEEN or TY ("one" is acceptable, but not "oneteen" or "onety"), some can be followed by TY but not TEEN ("twenty", but not "twenteen"), and some must be followed by either TEEN or TY ("thirteen" or "thirty", but not "thir").

```
SMALLNUM = TEEN1 | DIGIT1 <"TEEN2 | "TY (DIGIT2)>
CENTI = (SMALLNUM1) ("HUNDRED (("AND) SMALLNUM2))
NUMBER = (CENTI1) ("THOUSAND (("AND) CENTI2))
```

Figure II-7. PHRASE STRUCTURE PARTS OF NUMBER RULES

The CENTI rule allows numbers like 2235 to be said in various ways including "twenty two hundred and thirty five". The CENTI procedure blocks sequences like 4000 said as "forty hundred" and also

computes the numeric value of the CENTI phrase from the values of the constituents. The NUMBER rule can construct number phrases like "two thousand and one". The procedure blocks phrases like 8100 said as "eight thousand hundred" or as "one thousand and seventy one hundred". A language definition for text input might omit such restrictions on the grounds that no one would ever violate them in practice. However, as mentioned previously, difficulties in acoustic processing can cause the system to 'hear' almost anything, so the language definition must take advantage of every opportunity to block unacceptable phrases and downgrade unlikely ones.

In the remainder of this section, the descriptions of rules are typically limited to simple sketches like the preceding ones. However, to give a better indication of how the rules are actually written, one rule procedure, for SMALLNUM, will be discussed in detail. The SMALLNUM rule definition is given in Figure II-8. The procedure body is a conditional statement with four main cases that depend on the constituent structure.

In the first case, the SMALLNUM is a TEEN (TEEN1 in the structure declaration), a number from the lexical category including "ten", "eleven", and "twelve". The attributes NUM and NUMTYP are copied to the SMALLNUM phrase from the TEEN by the ^ATTRS statement. The second case for SMALLNUM occurs when the suffix TEEN is used (TEEN2 in the structure declaration). In this case, there are two statements to be performed (grouped together by square brackets and separated by a

```

RULE.DEF SMALLNUM  SMALLNUM = TEEN1 | DIGIT1 <"TEEN2 | "TY (DIGIT2)>;
  IF HAVE TEEN1 THEN ^ATTRS NUMTYP,NUM FROM TEEN1
  ELSE IF HAVE TEEN2 THEN
    [IF TEEN(DIGIT1) EQ "NO THEN F.REJECT(F.DIGTYP1),
     IF COMPLETE.NODE THEN NUM=NUM(DIGIT1)+10]
  ELSE IF HAVE TY THEN
    [IF TY(DIGIT1) EQ "NO THEN F.REJECT(F.DIGTYP2),
     IF HAVE DIGIT2 THEN
       [IF ALONE(DIGIT2) EQ "NO THEN F.REJECT(F.DIGTYP3),
        IF COMPLETE.NODE THEN NUM=NUM(DIGIT1)*10+NUM(DIGIT2)]
      ELSE IF OMIT DIGIT2 THEN
        [NUMTYP="DECADE2,
         IF COMPLETE.NODE THEN NUM=NUM(DIGIT1)*10]]
    ELSE IF HAVE DIGIT1 AND OMITALL(TEEN2,TY) THEN
      [IF ALONE(DIGIT1) EQ "NO THEN F.REJECT(F.DIGTYP4),
       NUM=NUM(DIGIT1)];

```

Figure II-8. SMALLNUM RULE DEFINITION

comma). The first statement looks at the TEEN attribute of DIGIT1 and blocks the phrase if the attribute is NO by performing F.REJECT(F.DIGTYP1). This blocks bad DIGIT TEEN sequences such as "oneteen". The second statement tests the flag named "COMPLETE.NODE" and, if the flag is true, sets the NUM attribute, which gives the numerical value of the phrase, to ten plus the NUM attribute of DIGIT1. (The Executive sets COMPLETE.NODE false when applying a rule with some of the constituents missing or for special tests.) The third main SMALLNUM case is executed when HAVE TY is true. The first statement checks the TY attribute of DIGIT1 and blocks the phrase if the attribute value is NO (eliminating phrases like "onety"). The second statement is another conditional depending on the phrase structure. If HAVE DIGIT2 is true, two statements are performed: a check that DIGIT2 can occur

without a suffix -- that is, ALONE(DIGIT2) is not NO -- and an assignment statement setting the NUM attribute to NUM of DIGIT2 plus ten times NUM of DIGIT1. If HAVE DIGIT2 is false but OMIT DIGIT2 is true, the NUMTYP attribute is set to DECADE2. This attribute is used in checks in the CENTI rule to block phrases like "forty hundred". The NUM of the SMALLNUM is then set to ten times the NUM of DIGIT1. The fourth and final case for the SMALLNUM procedure occurs when the phrase has DIGIT1 and omits both suffixes. The ALONE attribute is checked to block the phrase if DIGIT1 needs a suffix (for example, "thir-" is in the lexicon as a digit that needs a suffix). If that test is passed, the NUM attribute is copied from the digit.

The SMALLNUM composition rule illustrates several points. First, the use of options and alternatives in the phrase structure declaration makes it easy to specify the basic possibilities. Second, the rule procedure is organized as nested conditional statements depending on the particular phrase structure. Third, unwanted phrases are blocked by tests referring to constituent attributes. These tests are embedded in conditionals in a manner ensuring that as soon as the necessary constituents are available, the tests are made. The tests are independent of the presence or absence of other constituents and are also insensitive to the order in which the constituents are acquired. Finally, the conditionals testing the structure treat HAVE and OMIT for a particular constituent as separate possibilities. HAVE and OMIT can both be false for a missing constituent in certain cases, since the

Executive needs to apply the rules with incomplete sets of constituents. For example, if DIGIT1 and TY have been found for a SMALLNUM, then without waiting for a possible DIGIT2, the rule procedure will be executed to confirm that the DIGIT1 can occur with the TY suffix.

As an example composition rule, SMALLNUM accurately reflects the general form, but it is atypically simple. In contrast to the one-third page size of the SMALLNUM rule, the average rule length is about one page, and the biggest rule, for noun phrases, is almost three pages. Consequently, the following discussions are limited to sketches of rules rather than exhaustive, line-by-line documentation.

The phase structure declaration for the noun phrase rule is

```
NP = {"HOW.MANY | <DET | WHDET | "A" (NUMBER)}  
      ((CLASSIFIER) N ("PL) (PREPP)).
```

The noun, N, can be optionally preceded by a CLASSIFIER and followed by a plural suffix (PL) and a prepositional phrase (PREPP). At the front of the noun phrase, there can optionally be "how many" or an optional choice of DET, WHDET, or "a", followed by an optional number. This phrase structure allows many possibilities, some of which must be blocked by the NP procedure. For instance, "a" must be blocked if it occurs without a following number or noun. Other structures are blocked in certain cases depending on the attributes of constituent phrases. For example, the NUMBER cannot be leftmost if it begins with "hundred" or "thousand". On the other hand, if the NUMBER does not start with "hundred" or "thousand", it cannot be preceded by "a".

The NP procedure has a large number of statements related to the head noun, N. Several NP attributes are derived from corresponding N attributes, and many of the restrictions on possible NPs depend on properties of the N. For example, if N refers to an individual such as a particular company, ship, or country, it cannot be preceded by "how many", "a", WHDET, CLASSIFIER, or NUMBER, and the only preceding DET allowed is "the". There are also case semantics checks for the noun with a CLASSIFIER or a PREPP, and number agreement tests for the noun with several other NP constituents.

When the NP is complete, the procedure invokes routines to construct a semantic net representation of its meaning. If the NP has a definite determiner (like "the" or "that"), there are also calls on discourse routines to look for possible referents. The phrase is rejected if the semantic translation cannot be made or the discourse referents cannot be determined.

The phrase structure part of the verb phrase rule is

VP = V <<"SG | "PAST> NP | ("PPL) (PREPP)>.

The constituents are verb (V), singular suffix (SG), past tense suffix (PAST), object noun phrase (NP), passive suffix (PPL), and prepositional phrase (PREPP). The VP procedure checks various attributes of the verb and other constituents to block unwanted combinations such as a verb marked as active (like "have") followed by a passive marker (PPL). There are similar syntactic checks regarding tense and number. Case semantics checks ensure that the verb is compatible with the object and the prepositional phrase.

The prepositional phrase rule is one of the simplest. The phrase structure is just PREP NP. The PREPP procedure blocks the phrase if the NP is nominative case (like "we"), or if the preposition and the noun phrase do not go together semantically. The phrase is given a low rating if the NP is marked WH, or if the NP contains a NUMBER and the noun is not a unit or a relation. (For example, "of ten knots" is okay, but "of ten ships" is considered unlikely in view of the expected questions.)

The remaining rules are for the root category, S. The simplest S rule has the phrase structure HOW ADJ BE NP, illustrated by a sentence like "How fast is it?" The rule procedure checks the semantics of the adjective and the noun phrase for compatibility. Phrases are blocked if BE and NP do not agree in number, or if the NP is marked WH or accusative case ("us"). The phrase is given a low rating if the BE is past tense or the NP is indefinite and has a NUMBER. ("How fast are the ten ships?" is okay, but "How fast are ten ships?" is dubious.) The phrase is blocked if the NP is a unit or measure, or if the semantic translation fails for other reasons.

Another relatively simple S rule has the phrase structure S = (DO NP) VP. This rule handles imperatives and questions starting with a DO verb. If DO and NP are present, they must agree in number, the NP must not be marked as WH or accusative case, and the VP must not be imperative. If DO and NP are omitted, the VP must be imperative and must not be marked WH. In either case, the VP must not be marked as

singular, past, or passive, and the phrase is blocked if semantic translation fails.

The phrase structures for the last two S rules are

S = BE NP1 {NP2 | VP}, and

S = NP1 <(DO NP2) VP1 | BE {VP2 | NP3 | "THERE"}>.

These rules handle a variety of question types and elliptical sentences. Both make many tests concerning syntax, case semantics, and semantic translation. The procedure for the first rule is about one page long, and the second is about two pages in length.

In summary, the composition rules use the phrase structure declaration to give the basic constituent possibilities and use the procedure to block or downgrade unwanted combinations and upgrade expected ones. The procedures are organized as nested conditionals depending on the constituent structure. Simple syntactic tests are made first, followed by case semantics, and, finally, by semantic translation and discourse processing.

D. THE DEFINITION COMPILER

The Definition Compiler translates a language definition into a form for use by the Executive System. Data structures called 'records', containing a variety of information, are constructed for each category and rule. An important component of the category records is the list holding the lexical entries. Major components of the rule records are the phrase structure information and the rule procedure. This section describes the internal form of a language definition in detail and sketches the principal Definition Compiler algorithms.

1. CATEGORY RECORDS AND THE LEXICON

The global declarations for a language definition include a list of the categories. In addition to these declared categories, the Compiler creates special one-word lexical categories for each distinct literal used in the composition rules. These special categories are constructed so that the Executive does not have to treat literals as a separate case. For each category, declared or specially created, the Compiler constructs a record containing several components, the most important of which are the following:

- * A list of attributes for the category. These are derived from the global declarations section of the language definition.
- * A list of rule records for the rules that produce phrases of this category.
- * A list of categories that can occur as the leftmost terminal phrase in a phrase of this category. This

component, the next one, and the ones like it in the rule records, are used for "lookahead" by the Executive.

- * A list of categories that can occur as the rightmost terminal phrase in a phrase of this category.
- * A LISP function to set the attributes and factors of terminal phrases of this category. This function is created from the category procedure given in the language definition.
- * A list of lexical subcategory structures.

Each lexical subcategory structure is a list containing the name of the subcategory, the default attribute-value pairs for members of the subcategory, and the list of members. Each member is represented by a list with the word, its attribute-value pairs, and a back-pointer to the category record.

A lexical category declaration is compiled in a series of steps. It is first converted into a list structure, and the language CATEGORYFN, if any, is called to modify the definition. The subcategories are then compiled with the global WORDFN and the category WORDFN applied to each word definition before its attributes are stored.

In addition to the words declared in the language definition, the internal lexicon contains items called 'multiword lexical entries', or 'multiwords'. These items are treated as single units for acoustic processing but not for linguistic processing. For example, the phrases "of the" and "are the" are among the multiwords used in the speech system. The use of multiwords improves the acoustic performance by providing larger units for testing. However, the language definition

would become excessively complicated and lose linguistic generality if multiwords had to be treated as single words linguistically, so the Compiler and the Executive cooperate to hide the existence of the multiwords from the language definition. The Executive's treatment of multiwords is described in Chapter III, Section C.4. The Compiler's job is to add them to the lexicon so that a multiword phrase X starting with word A is included in all the lexical subcategories that include A. Thus, whenever the Executive considers A as a candidate word, X will be available for consideration also. Since the Executive can work in both directions in an utterance, the Compiler also adds multiwords Y that end in word B to all subcategories including B. The multiwords in the lexicon are marked to indicate whether they are to be considered in left-to-right tests (such as X) or right-to-left tests (such as Y).

2. RULE RECORDS, STRUCTURE GRAPHS, AND PROCEDURES

The Compiler creates a record for each rule containing, among other things, the following components:

- * A graph representing the phrase structure possibilities for the rule.
- * A list of categories that can occur as the leftmost terminal phrase in a phrase constructed by this rule.
- * A list of categories that can occur as the rightmost terminal phrase in a phrase constructed by this rule.
- * A LISP function created from the rule procedure.
- * A back-pointer to the category record for this rule.

Rule compilation proceeds in a series of steps: (1) the rule is translated into a list structure, (2) the language RULEFN, if any, is applied to modify the rule definition, (3) the phrase structure graph is created, and (4) the rule procedure is rewritten and compiled as a standard LISP function.

The phrase structure information is represented by an acyclic, directed graph. The arcs in the graph are labeled with either a category or NIL. Recall that literals are replaced by special one-word categories, so there is no need for a special kind of arc for literals. NIL arcs are introduced to deal with optional elements in the graph.* There is a unique starting, or "leftmost", point in the graph, and a unique ending, or "rightmost", point. A path is a series of arcs A_1, \dots, A_k , such that the end point of A_i is the starting point of A_{i+1} . It is a complete path if the starting point of A_1 is the leftmost point in the graph, and the endpoint of A_k is the rightmost point. Two points are connected if there is a path between them; by convention, a point is considered to be connected to itself by a zero-length path. The category labels along any complete path indicate a valid sequence of constituents for the rule. Figure II-9 shows a phrase structure declaration and its corresponding graph.

A phrase structure graph is stored as a collection of points and arcs. Each point is represented by lists of arcs coming in from the

* NIL arcs are somewhat like JUMP arcs in an augmented transition network (see Woods, 1970).

S = NP1 <(DO NP2) VP1 | BE {VP2 | NP3 | "THERE"}>

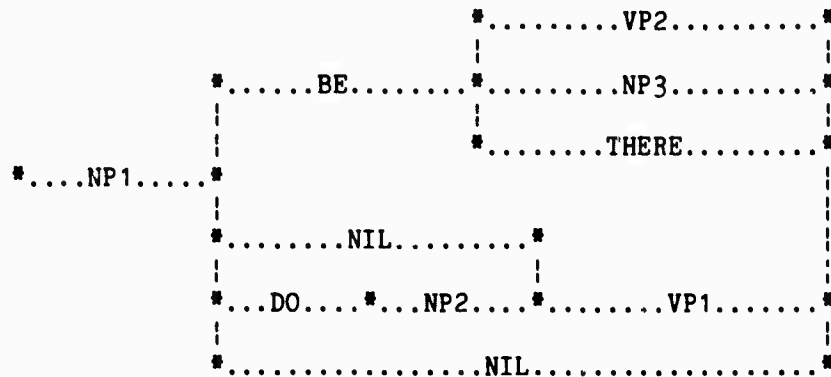


Figure II-9. A PHRASE STRUCTURE DECLARATION AND ITS CORRESPONDING GRAPH

left and arcs going out to the right. The arc lists for each direction from a point are divided into separate sections for category arcs and NIL arcs so that the Executive does not have to test each arc every time it is used to see which kind it is. Each arc is represented by a list containing its starting point, its ending point, its label (a category or NIL), an index number, and a table indicating other arcs in the graph that cannot occur in complete paths that contain this arc (for instance, the BE arc above cannot occur in complete paths with DO, NP2, or VP1 arcs or with either of the NIL arcs). Notice that this representation allows the Executive to search the graph in either direction from any arc or point.

In processing an utterance, the Executive tries to get a series of adjacent phrases corresponding to the category labels on a

complete path through the structure graph. As the subphrases of a phrase are acquired, they are stored in a constituent-array for the phrase in the position specified by the index number of the corresponding category arc. The constituent-array is initialized to contain NILs, so the Executive can check whether it has acquired a constituent for a particular category arc by a simple array reference using the arc index number. The "HAVE constituentname" expressions in the rule procedures are also implemented as constituent-array references (requiring only two instructions in the PDP-10 INTERLISP). NIL arcs are assigned index numbers larger than those for the category arcs in order to minimize the size of the constituent array.

Since the Executive is often concerned with the relative order of constituents, the category arc index numbers are assigned such that if the index of category arc A is less than the index of category arc B, either A is to the left of B or they are mutually exclusive. (Arcs are mutually exclusive if there is no complete path that includes both of them.) As an example, the order of the category arc index numbers in the graph shown above is NP1, BE, VP2, Nf3, THERE, DO, NP2, and VP1. The Executive takes advantage of the ordering of category arcs in many places. For instance, to find the first filled arc (an arc with an acquired constituent for the phrase under consideration) to the right of the arc with index number I, the Executive searches through the constituent-array for the first nonNIL entry at location I+1 or above. The arcs are also stored in an array according to their index numbers,

so if a phrase is found in location J of the constituent-array, the same index J can be used to access the corresponding arc in the graph. In this way, the Executive substitutes array scans for graph searches.

As constituents are acquired for a phrase, some arcs are filled, and others are blocked because they are mutually exclusive with the filled arcs. The Executive keeps track of which arcs are blocked by maintaining a bit table with each phrase. (The table is actually implemented as a single integer, thus limiting the total number of arcs in a graph to 36. This limit has not been a problem in practice.) Bit number I is turned on if and only if the arc with index number I is blocked. Part of the data stored for each arc is a bit table with the bits turned on for the arcs that are blocked by it. Whenever an arc is used, the Executive updates the bit table for the phrase by ORing it with the bit table for the arc. The Executive tests if an arc is blocked by checking the corresponding bit in the table (which takes seven instructions after the value of the arc index is loaded into a register). The "OMIT constituentname" expressions in the rule procedure also refer to the bit table. (The OMIT expression is compiled in only four instructions since the constituent index number is known at compile-time.) Similarly, OMITANY and OMITALL are implemented by creating a bit table for the constituents in question and ANDing it with the phrase bit table or its complement (taking a total of five instructions).

The operation of the Executive is simplified further by adding redundant NIL arcs to the graph so that it is never necessary to traverse two NIL arcs in a row. If two points, A and B, in the graph are connected by a path of two or more NIL arcs, but no single NIL arc connects them, the Compiler adds a new one to connect A and B directly. No redundant NIL arcs are added to the graph in Figure II-9, but in other cases, such as the NP rule, many are needed. Figure II-10 shows the NP graph before redundant NIL arcs are added. To this graph, the compiler adds five NIL arcs: (1) from the leftmost point to the point at the right of the NUMBER arc, (2) from the left of the NUMBER to the right of the CLASSIFIER, (3) from the leftmost point to the right of CLASSIFIER, (4) from the left of NUMBER to the rightmost point, and (5) from the left of PL to the rightmost point.

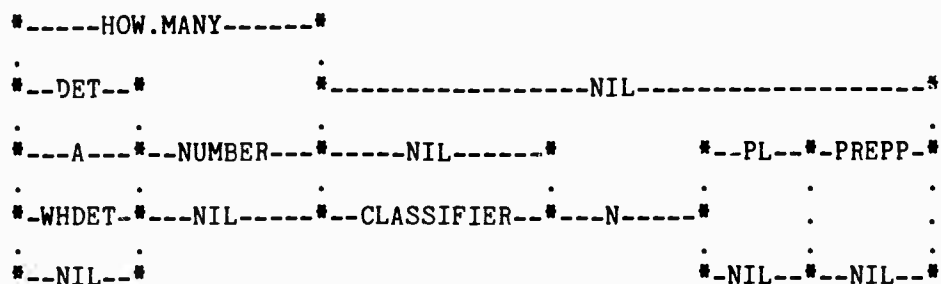


Figure II-10. NP GRAPH BEFORE ADDITION OF EXTRA NIL ARCS

The redundant arcs simplify the Executive algorithms by allowing iterative operations to replace recursive searches of arbitrarily long, and perhaps converging, paths of NIL arcs. For example, to check all the categories that can occur immediately to the

left of a given constituent, the Executive can fetch the point at the left of the arc for the constituent, check the incoming category arcs, and then for each incoming NIL arc, check the incoming category arcs for the point at the left of it. Because of the presence of redundant NIL arcs, this simple algorithm covers all the possibilities without duplication. For example, to the left of the N arc in the NP graph after the redundant NIL arcs have been added, there is an incoming category arc for CLASSIFIER and three incoming NIL arcs: one to NUMBER and HOW.MANY, a second to DET, WHDET, and A, and a third to the leftmost point in the graph.

The Executive can acquire constituents of a phrase in any order, not just left to right. Consequently, after each category arc is filled, tests are made to see if a complete path has been created. The tests succeed if there is a filled path from the left of the newly filled arc to the leftmost point in the graph and a filled path to the rightmost point. (A path is filled if all of its category arcs are filled.) NIL arcs are used in the search for filled paths if they are not marked as blocked. The Compiler makes this search more efficient by ordering the list of NIL arcs from each point so the Executive never needs to try more than one of them. The outgoing NIL arcs from a point are ordered such that if arc A precedes arc B in the list, no path from the endpoint of A leads to the endpoint of B. Similarly, the incoming arcs are ordered such that if A precedes B, no path leads from the starting point of B to the starting point of A. Basically, this means

putting the 'longest' arcs at the front of the lists. For example, the NIL arcs coming in to the left of the N in the NP graph are ordered such that the first goes to the leftmost point, the second goes to the point at the left of NUMBER, and the third goes to the point at the left of CLASSIFIER. With the NIL arcs ordered in this way, the search for a filled path only needs to consider the first unblocked NIL arc. If the first one fails to lead to a filled path, none of the following ones can possibly succeed. To prove this, assume to the contrary that A precedes B in a list of outgoing arcs, both are unblocked, a search to the right starting with A fails to lead to a filled path, but a search using B succeeds. The point at the right of B cannot be the rightmost point in the graph, since A leads to the rightmost point, and A is before B in the list. Thus, there must be a filled category arc immediately following B. However, this contradicts the hypothesis that A is unblocked since no path including A leads to the end of B where the filled arc begins. The proof for incoming NIL arcs is similar.

To review, the phrase structure declaration for a rule is translated into an acyclic, directed graph. The arcs are labeled with a category or NIL and are assigned index numbers reflecting their left-to-right order. Arcs and constituents are stored in parallel arrays ordered by arc index number. A bit table is stored with each phrase to record which arcs are blocked. Other bit tables are stored with each arc to indicate the other arcs that are mutually exclusive with it. Redundant NIL arcs are added to the graph so that paths do not need to

include two NIL arcs in a row. Finally, the NIL arcs from a point are ordered so that a search for a filled path can stop after considering the first unblocked NIL arc.

3 DETAILS OF RULE COMPILATION ALGORITHMS

The following paragraphs sketch the Compiler algorithms for translating the phrase structure declarations into their internal form. The translation begins with the creation of an initial graph. Recall from the formal syntax given previously that a phrase structure is a set of alternatives, each alternative is an ordered series of elements, and each element is either a category, a literal, an optional series, a set of alternatives, or an optional set of alternatives. To create an initial graph for such a phrase structure declaration, the Compiler first creates the leftmost and rightmost points, and then, for each top-level alternative, it creates a graph for the series of elements in the alternative, starting at the leftmost point and ending at the rightmost. To create a graph for a series of elements E_1, \dots, E_n , from point A to point B, the Compiler creates $n-1$ intermediate points P_1, \dots, P_{n-1} , and then creates graphs for E_1 from A to P_1 , for E_2 from P_1 to P_2 , ..., and for E_n from P_{n-1} to B. A graph for an element E from points A to B depends on what kind of element E is. If it is a category (or literal), a category arc from A to B is constructed. If it is a set of alternatives, a graph from A to B is created for each alternative series of elements. If it is an optional series, a graph from A to B for the

series is created, and then a NIL arc from A to B is added. Similarly, an optional set of alternatives is handled by creating the graphs for the alternatives and adding a parallel NIL arc.

The next step is to add the redundant NIL arcs. The Compiler keeps adding NIL arcs as long as it finds two in sequence between points A and B, and no single NIL arc joins A and B. After this process is complete, duplicate NIL arcs are deleted, as are any joining the leftmost point to the rightmost.

The lists of incoming and outgoing NIL arcs for each point are then ordered. The lists are sorted by exchanging arcs A and B until it is the case that if A precedes B, then, for incoming arcs, there is no path from the starting point of B to the starting point of A, or, for outgoing arcs, there is no path from the end point of A to the end point of B.

Arc index numbers are also assigned by a sorting procedure. If there are N categories and literals in the phrase structure declaration, the category arcs get numbers 1 to N, and the NIL arcs get numbers above N. The numbers for category arcs are sorted by exchanging the numbers for arcs A and B until it is the case that if the index for A is less than the index for B, then there is no path from the right point of B to the left point of A.

The final operation is to make a bit table for each arc A indicating the arcs that are blocked by the use of A. This table is

formed by turning on the bit for each other arc B such that no left-to-right path exists either from the right point of B to the left point of A or from the right point of A to the left point of B.

The arcs are stored in an array according to their index number, and the arc-array is stored as part of the rule record. Also stored in the rule record are the leftmost and rightmost points in the graph and the number of category arcs. This additional information could be derived from the array of arcs, but the Executive benefits by having it directly available.

After the phrase structure declaration is translated, the Compiler begins work on the rule procedure. The procedure statements dealing with attributes, factors, and constituent structure are rewritten as standard LISP statements that will work in the environment provided by the Executive. Before calling the rule procedure, the Executive sets up an environment containing: (1) the constituent-array for the phrase, (2) the bit table showing blocked arcs, (3) the array of attribute values, and (4) the array of factor values. The Compiler converts "HAVE constituentname" expressions to constituent-array references using the appropriate category arc index. Similarly, "OMIT constituentname" expressions are converted to a test of the appropriate bit in the bit table of blocked arcs. In both cases, the Compiler looks up the arc number corresponding to the constituent name, and a macro produces the required code.

References to attributes are converted to attribute-array references using as index the position of the attribute in the category attribute list. Factor references are converted to factor-array references using as index the position of the factor name in the list of factors for the rule. In both cases, the array indexes are constants known at compile-time, so efficient code is produced. For example, references to constituent attributes produce eight PDP-10 instructions to load an item from the constituent-array, and to give UNDEFINED if the item is NIL or, if it is not, get the attribute value from the attribute-array of the constituent.

The statement employed to abort a rule,
"F.REJECT(factorname)",
is converted to a call on a function named F.REJECT with two arguments: the name of the factor and the name of the rule procedure. The F.REJECT function calls a LISP subroutine ("RETFROM") to cause the rule function to return immediately with the value NIL as an indication of failure. The factor name is passed to F.REJECT as an aid to debugging. If a rule is rejecting an input that the definition writer intended it to accept, it is often because of a bug in a factor statement. The offending statement can be easily located by watching calls on F.REJECT from the rule procedure.

The last step in compiling a rule is to create an empty phrase for it, a phrase with no constituents. The phrase is saved with the rule record and used by the Executive in ways described in the next chapter.

4. LOOKAHEAD INFORMATION

Both category records and rule records contain lists of the categories that can occur as the leftmost or rightmost terminal phrases of that category or rule. This information is used by the Executive to 'look ahead' to avoid unnecessary work on a category or rule whose possible categories for boundary words do not intersect the categories of the word possibilities determined by acoustic tests. For example, before trying to construct a verb phrase starting at a particular location in the input, the Executive checks acoustic results for that location to ensure that the possibilities include at least one word that can occur as the leftmost word in a verb phrase. The next chapter contains more discussion of the use of lookahead by the Executive.

The lookahead lists are constructed in the following way. Each category is first added to its own list of possible leftmost and rightmost categories. Then, for each rule producing a phrase of category A and for each constituent of category B that can occur as the leftmost immediate constituent of the rule, the Compiler calls the procedure ADDLEFTCAT to add B to the list for A. If B is already on A's list, ADDLEFTCAT does nothing. Otherwise, it adds B and then propagates the addition in the following way: (1) for each category C that is a possible left category of B, ADDLEFTCAT calls itself recursively to add C to the list for A, and (2) for each category D that includes A as one of its possible left categories, ADDLEFTCAT calls itself recursively to add B to the list for D. A similar operation is performed for rightmost

category lists. When this process is completed for all the rules, the category lists are trimmed to eliminate categories with no lexical entries. The rule category lists are finally set to be the union of the lists for the categories that can occur as their leftmost or rightmost immediate constituents.

E. DISCUSSION

As described in the preceding sections, the Definition System consists of a metalanguage and a compiler. The metalanguage is designed to provide a means for integrating the contributions of a variety of knowledge sources while avoiding commitment to a particular overall control strategy. The basic approach in the metalanguage is to use augmented phrase structure (APS) rules in which a structure declaration gives the constituent possibilities and an associated procedure defines attributes and factors for phrases built by the rule. A major job of the Definition Compiler is to construct an internal representation of the definition for use by the Executive in processing sentences. Structure graphs are constructed by the Compiler from the phrase structure declarations, and LISP procedures are written and compiled to implement the rule procedures. The Compiler also builds an internal lexicon that includes special entries for 'multiwords.' Finally, lookahead information is computed and stored for categories and rules. In this section, we compare the Definition System to some alternative approaches that have been used in previous efforts.

The best known natural language understanding system is undoubtedly Winograd's SHRDLU (Winograd, 1971). The language definition system used in SHRDLU is called PROGRAMMAR, and, like the other components of SHRDLU, emphasizes a procedural approach to representing knowledge. A PROGRAMMAR program is designed for top-down, left-to-right sentence processing. The structural possibilities for the defined language are encoded in the control structure of the program rather than being declared separately in a form such as phrase structure rules. This method reflects a desire to encode a great deal of special-case knowledge to guide processing as an alternative to relying on a uniform but weak algorithm. The emphasis on special-case knowledge and close cooperation among different knowledge sources during sentence processing is a major contribution, but the particular method used in PROGRAMMAR makes it difficult to experiment with different overall control strategies. In earlier speech understanding work at SRI, we used a procedural approach in the PROGRAMMAR tradition (Walker, 1973a,b), but that approach was abandoned to allow freer experimentation. Our current approach retains the PROGRAMMAR emphasis on special-case knowledge and close cooperation of knowledge sources, but it eliminates from the language definition the commitment to a particular control strategy. Procedural representation is limited to attribute and factor information; the structural possibilities for the defined language are declared separately rather than being encoded in the control structure of a program. Thus, the use of APS rules attempts to keep the most valuable aspects of PROGRAMMAR's procedural representation while eliminating its constraints on system control options.

Another well-known approach is the use of augmented transition networks (ATNs) for language definition (Thorne, Bratley, and Dewar, 1968; Bobrow and Fraser, 1969; Woods, 1970). ATNs are extended versions of finite-state machines of automata theory. The first extension is to allow state transitions to depend on the successful execution of an entire network rather than being limited to testing a single word. By this extension, context-free languages can be handled. The second extension is to allow each arc to have an arbitrary condition associated with it that must be satisfied for the arc to be used in a transition. This extension gives ATNs the theoretical power of Turing machines.

ATNs have been used successfully in several large natural language understanding systems (such as LUNAR described in Woods, Kaplan, and Nash-Webber, 1972), and the approach is also used in the BBN speech-understanding system (see papers in the 1974 IEEE Symposium, Erman, 1974b). However, we prefer an approach based on augmenting phrase-structure rules rather than transition networks. One reason for this is a personal preference for reading and writing rules rather than ATN networks,* but a less subjective reason concerns the relative freedom from control strategy commitments. Recall that our objection to Winograd's PROGRAMMAR approach centered around its commitment to a particular control strategy. However, as Winograd notes, PROGRAMMAR

* This preference appears to be shared by some users of ATNs. In a recent report, BBN comments that to document its grammar it has used a "semi-BNF" notation "which indicates much more clearly than the grammar listing what sorts of sentences are accepted by the grammar" (Woods et al., 1976a, p.10).

programs and ATNs "are just two different ways of talking about doing exactly the same thing" (Winograd, 1971, p.201). An ATN is conceptually a description of a nondeterministic machine. To process a sentence, the machine moves through a series of states specified by the structure and conditions of the ATN. The order of transitions is fixed, at least conceptually, by the left-to-right scan of the sentence, so tests and actions associated with arcs at the right of a network make use of information from previous arcs to the left. This left-to-right assumption affects the writing of augments on the arcs, but it is not an absolute barrier to the use of other control strategies. Unlike a PROGRAMMAR program, an ATN does separate the basic structure (i.e., the network) from the augments (the tests and actions associated with the arcs), so it is possible to use ATNs with non-left-to-right control strategies. The method for doing this depends on recognizing augments that use contextual information and delaying their execution until the necessary information is available (see Bates, 1975). Our APS rules avoid control commitments, conceptual or otherwise, by putting the augments in a single procedure rather than spreading them over a network; if a test or action uses information from several constituents, it is embedded in conditional statements that check for the relevant structure. The augments are thus organized in a way that avoids the left-to-right bias of ATNs. Although that bias can be circumvented, we prefer to use a representation that eliminates it rather than forcing the Executive to try to work around it.

As a final comment regarding ATNs, note that our internal representation for rules is like an augmented transition network with the augments collected in a single procedure. As mentioned above, the procedures are organized to avoid control strategy commitments, and the networks provide the Executive with explicit knowledge of the basic structural possibilities of the language in a form that is easy to use. The structure information is heavily used by the Executive in making predictions and constructing phrases (as discussed in Chapter III). Thus, in rejecting ATNs, we are not rejecting the value of networks as a representation. Instead, by changing the manner of adding augments and by constructing the networks automatically, we retain their internal efficiencies, and we also get an opportunity to optimize the network format during compilation (as in the addition of extra NIL arcs and the reordering of NIL arcs).

Our preference for APS rules over more procedural methods such as PROGRAMMAR and ATNs is shared by others. Such a preference appears, in fact, in early work on compilers for programming languages. The first programming language compilers were completely procedural; the definition of the language was embedded (lost) in the control structure of the compiler. In reaction to the obscurity of this method, "syntax-directed compiling" was developed by Irons and others (see, for instance, Irons, 1961; and Cheatham and Sattley, 1964). The developers of the new method were explicitly concerned with separating the two functions of defining the language and translating it, functions which

are merged in procedural approaches (see opening comments in Irons, 1961). Irons used a version of APS rules to state the syntax and semantics of a programming language. Each phrase structure rule had an associated semantic definition to form a 'translation' for a phrase constructed by the rule. The translation was the only attribute of the phrase and was formed from the translations of the constituents. Irons implemented a general translator program to operate on such language definitions and demonstrated the usefulness of the approach by developing an ALGOL 60 compiler.

Irons' technique of using APS rules for programming languages was extended by Knuth in a paper on the "semantics of context-free languages" (Knuth, 1968). Knuth's first extension was to allow an arbitrary number of attributes with each phrase. A set of attribute-defining functions was associated with each phrase structure rule rather than the single translation function of Irons. The second and more significant extension was to allow both 'synthesized' and 'inherited' attributes. Synthesized attributes of a phrase are defined solely in terms of attributes of the constituents of the phrase. Irons' translation attributes and our rule attributes are of this type. Inherited attributes of a phrase are defined by functions associated with phrases that include it as a constituent. In other words, these attributes are 'inherited' from the context rather than being 'synthesized' from information local to the phrase. In this system, there is a danger of circular definitions of attributes (such as

inherited attribute A depending on attribute B, which is in turn synthesized by a function with A as an argument), but such circular definitions can at least be detected automatically by an algorithm sketched in Knuth's paper.*

Knuth points out that inherited attributes do not provide greater theoretical power since "synthesized attributes alone are (in principle) sufficient to define any function of a derivation tree" (Knuth, 1968, p.142). However, he claims that in practice the use of both kinds of attributes can lead to important simplifications producing more "natural" definitions. To support this claim, he gives a small language definition that makes use of both synthesized and inherited attributes. The inherited attributes are used for operations such as testing the agreement between the declaration and the use of variables. Knuth comments that "in general, inherited attributes are useful when part of the meaning of some construction is determined by the context in which the construction appears" (Knuth, 1968, p.142).

Although Knuth's discussion is limited to programming languages, his system of inherited and synthesized attributes appears attractive for use in natural language processing. In fact, it has been used for semantic translation in the REQUEST system, which is an experimental question-answering system based on a transformational grammar of English

* However, the problem of determining whether the grammar avoids circularity in all possible instances is very difficult computationally. See Jazayeri, Ogden, and Rounds (1975) for a proof that any correct algorithm for solving this problem must require time that grows exponentially with the size of the grammar.

(see Petrick, 1973, 1976). In REQUEST, an input is parsed according to a surface structure context-free grammar, the surface tree is converted to a deep-structure tree by reversed transformations, and the deep-structure tree is mapped into a "logical representation" by Knuth's translation technique, using both synthesized and inherited attributes.

Faced with Knuth's claims supported by the example of REQUEST, we must explain our decision restricting the APS rules to use only synthesized attributes. In this case, as with our rejection of PROGRAMMAR-like procedural representations, the primary motivation is the desire to free the language definition from features that would excessively constrain the options for the Executive. Knuth states that his approach does not depend on any particular form of syntactic analysis. This is certainly true if the attributes are not to be computed until after a complete derivation tree is constructed, but we cannot afford to force the Executive to find complete context-free parses before drawing on attribute and factor information. The Executive must be free to use such information during sentence processing to limit and direct its efforts. Furthermore, inherited attributes make it difficult to share a phrase among several competing contexts. Such sharing is particularly important with speech understanding since acoustic uncertainty leads to a large number of alternative contexts. Inherited attributes are context dependent, so they, and all other attributes depending on them, would have to be duplicated for each context. Thus, we have restricted ourselves to

using only synthesized attributes because we cannot delay the use of augments until a complete parse is found, and we cannot afford to duplicate attribute and factor information for each context. The restriction to synthesized attributes and factors provides important flexibility in the Executive, and, to date, it has not been an impediment to the development of the SRI language definition.

A variety of computer systems for processing natural language have used some form of APS rules (for example, Sager and Grishman, 1975; Hobbs, 1974; Heidorn, 1975, Pratt, 1975, Landsbergen, 1976). The first was the Linguistic String Parser implemented at New York University under Sager in 1964-1965. The system has been redesigned and reimplemented since then, but it has continued to use a two component grammar: context-free rules defining the broad construction patterns of sentences, and restrictions covering detailed constraints. There is an emphasis on restrictions (corresponding to our Boolean factors), but the system does allow attributes to be set for nodes in the parse tree. In contrast with our approach, the restrictions for a rule are not organized into a single procedure. Instead, the approach foreshadows ATNs by associating restrictions with particular positions in the rules. As with ATNs, the positioning of restrictions assumes left-to-right sentence processing. For example, in a rule $A=BC$, a restriction might be positioned between B and C so it would be executed after the B phrase was acquired and before the C was tried. Finally, some of the 'restrictions' are really optimizations for the top-down back-up parser,

so there is some blurring of the distinction between the language definition and the control strategy for applying the definition. From our standpoint, this blurring and the left-to-right bias caused by positioning restrictions within rules are both shortcomings of the approach. However, the successful application of the approach to produce a grammar of very wide scope is evidence for the value of using APS rules for natural language.

Other APS systems for natural language processing have avoided the shortcomings mentioned above. For example, PHLIQA1 uses APS rules each with a single procedure for augments to translate from English to the first of several levels of semantic translation (Landsbergen, 1976; Scha, 1976). Our work was influenced by PHLIQA1 and differs mainly in allowing alternatives and options in structure declarations and in providing for nonBoolean factors in addition to Boolean restrictions. These additions are especially important in a system for speech understanding: the alternatives and options reduce the number of rules and hence decrease the storage requirements of the Executive, and the nonBoolean factors are of use in setting Executive priorities.

To summarize the preceding discussion, our Definition System continues a long-established line of systems using APS rules. We share with earlier developers, such as Irons, the desire to keep the language definition free of control strategy commitments in order to make the definition simpler and to allow greater flexibility in experimenting with different system designs. Our system differs from previous ones in

having broader phrase structure declaration capabilities and in allowing nonBoolean factors.

Up to this point the discussion has focused on the metalanguage and the choice of APS rules as a representation. The other major component of the Definition System is the Compiler. The Compiler creates an internal representation of a language definition for use by the Executive in sentence processing. The internal representation has several features that differentiate it from those used in previous systems. The networks representing the phrase structure declarations are reminiscent of ATNs or charts (see papers by Kay, 1973; Kaplan, 1973a) but they are distinguished from those systems by the presence of extra NIL arcs and the ordering of NIL arcs, both changes that contribute to Executive efficiency. Other distinctive features of the internal representation of the language are also concerned with efficiency of Executive operations. These features are (1) the use of parallel arrays for structure graph arcs and phrase constituents, with entries ordered to reflect the left-to-right structural possibilities, (2) the use of bit tables to keep track of blocked arcs and mutually exclusive arcs, (3) the translation of rule procedures into compiled LISP functions employing in-line instructions for efficient operations on attributes and factors and quick tests of constituent structure, and (4) the construction of left and right lookahead information for both rules and categories. The internal representation is tied to the design of the Executive, so further discussion of the representation is deferred to Chapter III.

III THE EXECUTIVE SYSTEM

Prepared by William H. Paxton

CONTENTS:

- A. Introduction
- B. Parse Net
- C. Overview of the Executive
 - 1. Predict Task and Word Task
 - 2. Setting Priorities
 - 3. Starting the Task Cycle
 - 4. Stopping the Task Cycle
- D. Details of the Executive
 - 1. Word Task
 - a. Getting a Word and Creating a Terminal Phrase
 - b. Distributing a Phrase to Consumers
 - 2. Adding a Constituent to a Consumer
 - a. Preliminary Tests
 - b. Create Complete Nonterminal Phrase
 - c. Create Partially Filled Nonterminal Phrase
 - 3. Predict Task
 - a. Create Subnet
 - b. Assign Ratings
 - c. Cleanup
 - d. Dead Phrases and Predictions
 - 4. Multiword Lexical Entries
 - 5. Priority Setting
 - a. Factors
 - b. Phrase Scores
 - c. Phrase Ratings
 - d. Relation to Executive Tasks
 - 6. Adjusting Priorities and Focus by Inhibition
- E. Discussion
 - 1. Review
 - 2. CMU: HARPY and HEARSAY-II
 - 3. BBN: SPEECHLIS and HWIM
 - 4. Earlier SRI Systems

A. INTRODUCTION

This chapter discusses the Executive System. The Executive in the speech understanding system has three main responsibilities: (1) it coordinates the work of the other components of the system by calling acoustic processes and applying language definition procedures, (2) it assigns priorities to the various tasks in the system, and (3) it organizes hypotheses and results so that information is shared and duplication of effort is avoided. In other words, the Executive carries out the functions of integrating and controlling the system components. Experimental results, to be discussed in Chapter IV, show that the manner in which the Executive performs these functions has a large effect on the overall performance of the system. For example, different techniques for setting priorities result in significant differences in average accuracy and runtime.

In processing an utterance, the Executive performs a series of tasks to find words in the speech signal and to organize them into phrases of the input language with the ultimate goal of creating a root category phrase that spans the input. Thus, because we have designed the speech understanding system with the language definition as the primary mechanism for specifying knowledge source interactions, the Executive does the job of a parser in fulfilling its responsibilities for system integration and control. We might have divided the Executive box in the system diagram (see Chapter I, Figure I-2) into two boxes, perhaps calling one 'Control' and the other 'Parser', and then made

favorable comments about the modularity of our approach. However, that would belie the extent to which the parsing operations of the Executive have been shaped to serve its integration and control functions and would also fail to reflect the central place in the system we have given to the language definition -- system components are controlled via the language definition, so it is not accidental that the Executive does the parsing. Consequently, our system diagram has a single box for the Executive rather than two boxes for Control and Parser, and this chapter deals with both system control strategy and parsing.

The following sections contain (1) a description of the main Executive data structure, called the 'parse net', (2) an overview of the Executive in sufficient detail to allow the reader to understand both the discussion in the last section of this chapter and the experimental results covered in Chapter IV, (3) a complete description of the Executive, and (4) a discussion comparing this approach to several others and sketching its evolution. This chapter presumes familiarity with the internal representation of the language definition as described in Sections D.1 and D.2 of Chapter II.

B. PARSE NET

The 'parse net' is the principal data structure built by the Executive.* This section describes the form and content of the parse net; following sections describe the procedures that operate on it. Nodes in the parse net are either 'phrases' or 'predictions.' Phrases correspond to words or composition rules from the language definition. Predictions are for particular categories of phrases at particular input locations. 'Terminal' phrases contain a single word and are formed when words are acquired by acoustic tests. 'Nonterminal' phrases are formed when a language definition rule is applied to a set of constituent phrases. If there are no unfilled, unblocked category arcs in the phrase's structure graph, the phrase is called 'complete' (for a description of structure graphs, see Section D.2 in Chapter IV). Otherwise, more constituents can be added, so the phrase is called 'incomplete.' A complete phrase that is formed by adding missing constituents to an incomplete phrase P is called a 'completion' of P. Predictions are made as part of the process of acquiring constituents to fill category arcs in incomplete, nonterminal phrases. An incomplete phrase is called 'empty' if none of its category arcs are filled. In this terminology, the parse net of predictions and phrases holds intermediate hypotheses and results while completions of empty, root-category phrases are constructed. Such complete root-category phrases with their attributes and factors are called 'interpretations' of the input.

* The design of the parse net was inspired by Kaplan's multiprocessing consumer-producer approach (Kaplan, 1973b).

Phrases and predictions have time specifications indicating their position in the input. By analogy with written text, beginning and ending times are referred to as left and right, respectively. The times for terminal phrases are provided as part of the output of the word recognition routines. The times for nonterminal phrases come from the leftmost and rightmost constituents, if those constituents have been acquired for the phrase. An incomplete nonterminal phrase that is missing its boundary constituents has its times either 'fixed' or 'unfixed'. Fixed times for a phrase P are either boundaries of the utterance or times from complete phrases that might be adjacent to completions of P. A fixed right time for a phrase P constrains possible rightmost constituents of P to end at or near the specified position. In contrast, an unfixed right time for a phrase means that there are no constraints on the ending time of possible rightmost constituents. Fixed or unfixed left times have similar results. Predictions also have fixed or unfixed times that determine which phrases fulfill them. For example, if a prediction has a fixed left time of 50, a phrase fulfilling the prediction must start at or near 50. (The details of how near is near enough are discussed later.)

Other information saved with each phrase includes an array of attributes. For terminal phrases, the attributes come from the lexical entry for the word or are computed by the category procedure. For nonterminal phrases, the rule procedure computes attributes of the phrase from constituent attributes. Terminal phrases have a pointer to

the lexical entry that was used to construct them, and nonterminal phrases include a pointer to their composition rule, a list of constituent phrases, and a bit table showing the blocked arcs in the structure graph.

Each prediction in the parse net is for phrases of a particular category that meet particular time requirements. Stored with the prediction record, in addition to the category and times, are the following lists:

- * Instances -- Complete phrases of the predicted category that meet the time requirements. These phrases fulfill the prediction.
- * Word sets -- Sets containing words from the predicted category which, if accepted by acoustic tests, can be used to construct terminal phrases fulfilling the prediction.
- * Consumers -- Incomplete phrases that can have a phrase that fulfills the prediction added to them as a new constituent. Thus, the phrases on this list can 'consume' instances of the prediction.
- * Producers -- Incomplete phrases whose completions could be instances of this prediction. In other words, these phrases can 'produce' instances for the prediction.

A prediction thus serves as an intermediary between two sets of incomplete phrases: consumer phrases that are all missing a constituent of the predicted category at the predicted location in the input, and producer phrases that all might supply the missing constituents. Note that a phrase can be a producer for one prediction and a consumer for another. Thus, 'producers' and 'consumers' are not names for distinct classes of phrases, but instead are names reflecting structural

relations in the parse net. The full set of producer-consumer connections in the parse net make explicit the different sentential contexts for each phrase. This contextual information is used by the Executive in setting priorities and in lookahead. These operations and the operations that construct the parse net are sketched in the next section.

C. OVERVIEW OF THE EXECUTIVE

The Executive carries out a series of tasks adding predictions and phrases to the parse net. There are two main types of tasks: the predict task, which operates in a top-down manner, and the word task, which operates in a bottom-up manner.* This section sketches these tasks and briefly describes how the task priorities are established and how the series of tasks is started and stopped. The level of detail in the descriptions is minimal but adequate to provide the reader with the prerequisites for understanding both the discussion in Section E and the experiments reported in Chapter IV. For the reader who wants a detailed description of the Executive, this section provides an introduction that should make the details given later easier to understand.

* See Aho and Ullman (1972) for a discussion of top-down and bottom-up parsing strategies.

1. PREDICT TASK AND WORK TASK

Figure III-1 shows the basic outline of the two main types of Executive tasks. The predict task takes incomplete phrases and adds a subnet of predictions and phrases to the parse net. The creation of predictions for categories with lexical entries causes the word task to be scheduled. Performing the word task entails getting an accepted word (one that has passed the acoustic tests), constructing a terminal phrase, and distributing it to consumers in the parse net. Adding the phrase to a consumer can result in a complete phrase P, in which case P is also distributed to consumers, or an incomplete phrase Q, in which case the predict task is scheduled to make predictions for constituents that can be added to Q. The link in Figure III-1 from the cleanup stage of the predict task to the add-constituent-to-consumer operation reflects the possibility of an old prediction with instances acquiring a new consumer.

Both tasks are guided by lookahead; in other words, they avoid unnecessary operations by using information about the acoustically possible adjacent words. For example, if acoustic tests show that there are no adjectives starting to the right of a phrase P, then no structures are built using P that would require an adjective to its right. Both tasks can also work either left-to-right through an input or bidirectionally from words selected at arbitrary positions within an utterance. The system is designed to allow constituents of phrases to be added in any order, so experimentation with a variety of control

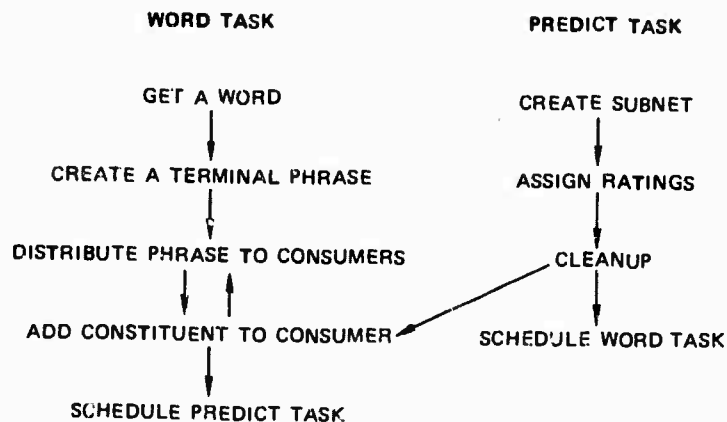


FIGURE III-1 EXECUTIVE TASKS

strategies has been possible. Most importantly from the system-control standpoint, each task does a limited amount of processing and then stops after scheduling further operations for later. The scheduling does not specify a particular time for a future operation, but instead gives the operation a certain priority. The operation is performed when it becomes top priority. This organization allows the Executive to control the overall activity of the system by setting task priorities.

We have experimented with two versions of the Executive tasks that differ in where the acoustic tests are performed. In the first version, called 'mapping one at a time', a word is tested as the first step of the word task, and only if the word passes the test is a

terminal phrase created. In this method, word priorities are primarily determined by consumer ratings (as described below). With the second method, called 'mapping all at once', all the word tests at a particular input location are performed before predictions are made at that location. The word priorities are then influenced by the results of the acoustic tests in addition to the consumer ratings. When a word becomes top priority, the word task goes directly to the step of creating a terminal phrase. Mapping all at once causes the system to test more words per location but yields better priorities since experimental results indicate that true hits tend to get higher scores than false alarms. The choice between mapping one at a time or mapping all at once is explored in the experiments reported in the next chapter.

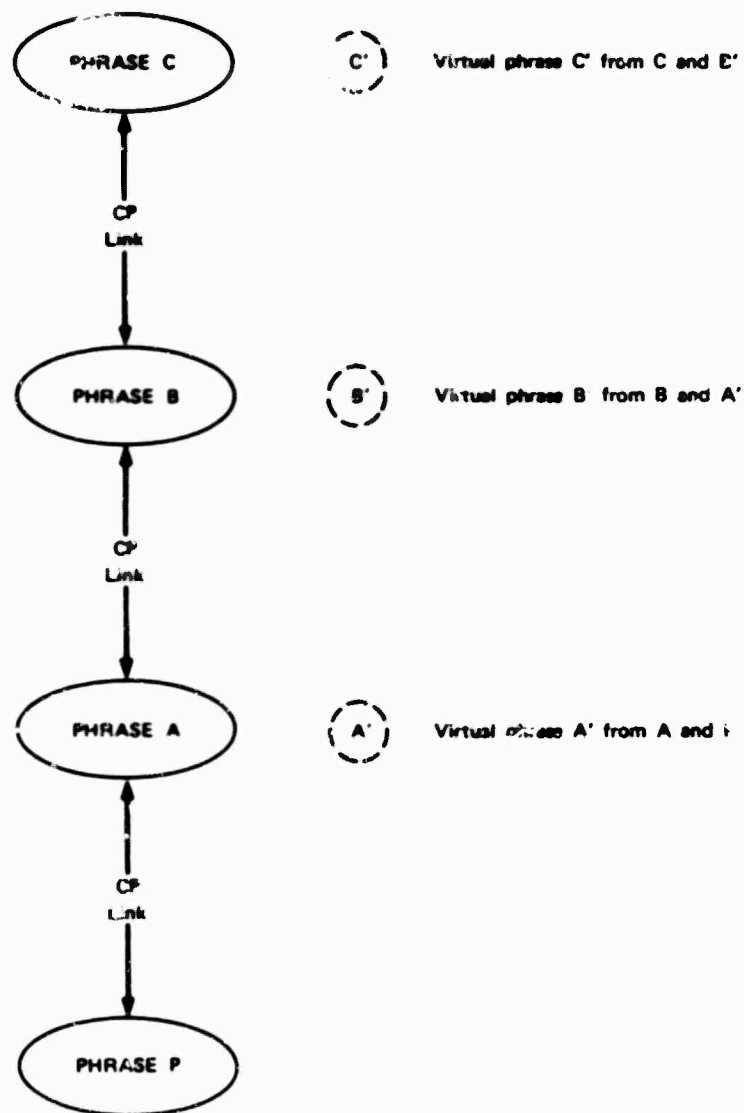
2. SETTING PRIORITIES

The fundamental data for priority setting are the word scores provided by the acoustic mapper and the factors computed by the language definition procedures. Mapper scores indicate how well a word matches the input signal at a particular location in the input. Language definition factors reflect likelihood judgments from syntactic, semantic, and discourse sources of knowledge. The 'score' for a phrase combines mapper scores, language factors, and, for a nonterminal phrase, the scores of its constituents. The score is thus a local, context-free piece of information about how 'good' the phrase is. The score may reflect global data such as a discourse model, but it does not depend on

possible sentential contexts for the phrase. In contrast, the 'rating' of a phrase does depend on the other phrases in which it may be embedded to form a sentence. The rating of a phrase P is intended to provide an estimate of the best score for an interpretation that can be constructed using a completion of P. If P is itself a root category phrase, its score determines its rating directly. Otherwise, the rating for P is determined by reference to the consumers for P in the parse net. (The organization of the Executive guarantees that all non-root-category phrases have at least one consumer.) We have experimented with two techniques for using the consumer context in setting phrase ratings. In one method, the rating with respect to a particular consumer is formed by adding the phrase score and the consumer rating. (Whenever possible, ratings are assigned top-down in the parse net so that consumer ratings are directly available for use in this process.) The phrase rating is then the maximum rating with respect to any of its consumers. This method is fast, but it leaves the rating unaffected by the consumer restrictions that are expressed in rule procedures rather than in structure declarations. A phrase may satisfy the structural requirements of a consumer C but still be incompatible with C because of constraints encoded in C's factor statements. For example, if the only sentential context being considered is "Is it owned by --", the structural requirements will be satisfied by any noun phrase, but semantic factors will restrict the alternatives to possible owners.

The second method for setting phrase ratings takes into account the procedural information in the rules by exploring the paths in the parse net that show how a phrase might be used and executing the corresponding procedures to gather attribute and factor information. Each producer-consumer path from a phrase P to a root category phrase reflects a way of constructing an interpretation using P. To calculate a rating for P with respect to such a complete path, temporary structures called 'virtual phrases' are built. For example, assume A is a consumer for P, B is a consumer for A, and C is a root-category consumer for B (see Figure III-2). The virtual phrase A' is formed by placing P in the appropriate empty constituent position in A. The attributes and score of A' indicate possible completions of A-plus-P. The virtual phrase B' is constructed by adding A' to B, and C' is constructed by adding B' to C. By assumption, C' is a root category phrase, so the score of C' determines the rating of P with respect to the consumer path A-B-C. Various paths from P are formed in this way, and the rating for P is its best rating with respect to any of the constructed paths.

To reduce the cost of rating alternatives by this method, a heuristic search is made in the parse net for a near optimal path rather than exhaustively trying all possible paths. The heuristic exploits the fact that, typically, when a phrase is being rated the higher level phrases that form its context have already been rated. (The parse net is initialized so that a context of previously rated phrases exists even



CP link is an indirect link between a consumer and a producer via an intermediate prediction.

SA-3804-4

FIGURE III-2 A CONSUMER PATH

when the system is doing bottom-up processing.) These prior ratings provide important heuristic information. The object is to find the path giving the best score, so the paths with the highest prior rating are explored first. When a complete path is found, one that leads to a root-category phrase, the score for that path sets a lower bound on the rating. This lower bound is used to prune paths whose prior ratings are low enough to suggest that they are unlikely to produce a rating higher than the lower bound already established.

This method takes more computation per rating assignment than the first one, but it should produce better phrase ratings since it gathers more information in forming them. Experimental results reported in the next chapter indicate that the extra effort spent in the second method is worthwhile; it leads to better system performance in both accuracy and runtime.

Phrase ratings are used to determine task priorities. The priority of the predict task comes from the highest rating of any phrase scheduled to make predictions. When the predict task is executed, it creates predictions and phrases only at the time and direction (left or right) that are determined by the best phrase scheduled to make predictions. Similarly, the priority of the word task is equal to the highest rating for any predicted word (the word rating is the rating of the terminal phrase that could be constructed from the word). When the word task is performed, it only operates on the highest rated word.

In the case where task priorities are directly determined by ratings, the control strategy is described as 'best-first'. We have experimented with modifying priorities to implement other control strategies in addition to best-first. In particular, we have tried a method we call 'focus by inhibition' in which high scoring words are selected from the best phrases for the predict-task, and tasks that cannot use those words are inhibited by having their priorities lowered. The selected words are the focus of attention for the system in this method and are described as 'the focus' or as 'in focus'. A phrase conflicts with the focus if it contains a nonfocus word that overlaps some focus word. The tasks that would try to complete such phrases have their priorities lowered. The priority reduction causes the system to be biased against working to complete phrases that conflict with the focus. If a task for a phrase P that is in conflict with the focus manages to overcome the system bias against it to become the task with the highest priority, the system shifts to a new focus by removing the words from focus that conflict with P and adding new words to focus from P.

The technique of focus by inhibition is motivated by a desire to reduce the thrashing among closely rated alternatives that can happen with a best-first strategy. Thrashing is reduced with focus by inhibition because the best phrase inhibits its competition and thus keeps the system's attention focused on fulfilling its predictions. The inhibition is a relatively small decrease in priority, so the bias can

be overcome. Therefore, focus by inhibition does allow the system to recover from selecting an incorrect word for focus. However, if the system focuses on incorrect words too often, the net effect of the priority changes can be harmful rather than helpful. Experimental results showed that selecting incorrect words was in fact a serious problem for focus by inhibition, and, as a result, overall performance was not improved by this technique. Although this particular attempt to improve performance by adjusting priorities did not succeed, the basic approach still merits further study. As a method for adjusting priorities, it provides simple answers to how, when, and why to focus attention, while still maintaining the completeness of the control strategy. (It does not discard alternatives, it simply revises their priorities.) Better success at selecting hits rather than false alarms for focus could result in a focus by inhibition that improved performance.

3. STARTING THE TASK CYCLE

The Executive starts processing an utterance with an initial parse net already in existence. If the system is using a left-to-right control strategy, the initial net contains: (1) for each root category rule, an empty phrase with times fixed at the beginning and end of the utterance, and (2) for each category that can occur at the left of an input, a prediction with its associated empty producers, all with their left times fixed at the beginning of the utterance and their right times

unfixed. Each phrase P in the initial parse net is connected as a consumer to the predictions for categories of phrases that can occur in P as leftmost immediate constituents. The Executive task cycle starts by scheduling the word task for the predictions in the initial net. This task will find a word at the start of the utterance and the interplay of word task and predict task will start.

As an alternative to left-to-right processing, the system can use 'island driving' in which phrases are constructed bidirectionally around 'island' words selected at arbitrary locations in the input.* The motivation for island driving is that it allows the system to begin processing an utterance where it is most confident that it has found a correct word. It can use that word to provide contextual guidance in processing other parts of the utterance where it is less confident. In contrast, left-to-right processing must start at the beginning of the utterance even if the system is not confident about any of the words there.

For island driving, the initial parse net contains: empty root category phrases with times fixed at the beginning and end of the utterance, and, for each category in the language, a 'monitor' (which is a special kind of prediction) with its associated empty producers, all with both times unfixed. Each phrase P in the initial parse net is

* Island-driving is derived from Miller's 'locally organized parsing' based on 'islands of reliability' (see Miller, 1973). For examples of its use in speech understanding systems, see Ritea, 1974 and Bates, 1975.

connected as a consumer to the monitors for categories of phrases that can be added as immediate constituents of P. The task cycle starts with the selection of an island word according to a criterion combining the word's mapper score and its estimated likelihood of being a false alarm. The word task is performed for this island word to create a terminal phrase. The terminal phrase is then passed to the distribute-phrase procedure. For island driving, this procedure is modified so that if the phrase being distributed does not fulfill any predictions, it is given to the consumers for the monitor of its category. In general, this operation can construct incomplete phrases that lead to predictions on either side of the island word. After the first island word is distributed, the Executive schedules a task to select a second island word in case the first one fails to lead to highly rated phrases. If this task is performed and starts a second island, it will reschedule itself to try a third in case the second runs into trouble. In this manner, a number of islands can be worked on simultaneously. The effect of island driving on system performance is a topic of Chapter IV, Section E.

4. STOPPING THE TASK CYCLE

After each execution of a task, the Executive checks several parameters to see if it should stop the task cycle. For instance, the Executive keeps track of the amount of storage in use and stops before the available storage is exhausted. Another stopping criterion is the

difference between the priority of the best remaining task and the processing time already used for the utterance. The Executive stops if the value of this criterion falls below a certain threshold. The threshold is initialized to a low value, but whenever an interpretation is constructed, the threshold can be raised so that the system will not spend much more time looking for other interpretations unless the priorities are high. When the Executive decides to stop, it calls the language RESPONSEFN function. This function is also called whenever an interpretation is constructed, and it stores the interpretations and manipulates the priority-minus-processing-time threshold. When the Executive tells the RESPONSEFN that it is time to stop, the function initiates question answering using the highest rated interpretation it has.

This concludes the overview of the Executive. The reader has an option at this point of skipping ahead to the discussion section at the end of this chapter or to Chapter IV (which deals with a series of experiments concerning system performance and the Executive) before going on to the following detailed description of the Executive.

D. DETAILS OF THE EXECUTIVE

This section gives a detailed explanation of the Executive. The topics covered are the word task, adding a constituent to a consumer, the predict task, multiword lexical entries, and priority setting. To make the following descriptions complete, there is some repetition of information covered in the overview.

1. WORD TASK

The major operations in the word task are to acquire a word that is accepted by the acoustic tests, create a terminal phrase for it, distribute the phrase to consumers, and schedule the predict task for any incomplete phrases that result. Predicted words are organized into 'word sets'. Each word set has a list of words from some lexical subcategory, time specifications like those for a prediction, and priority information. Word sets are typically created during the final step of a predict task, but they are also created at the start of a left-to-right parse. Like a prediction, a word set has one fixed time and one unfixed time.* The creation of a word set begins by finding the subset of the lexicon that is worth considering at the fixed time. If the system is using the mapping-all-at-once control strategy, this subset contains the words actually accepted by the mapper at or near the time. Otherwise, the subset is created by a special acoustic process

* Word sets for the root-category are exceptional in that both of their times are fixed. The algorithms take care of these as special cases.

called lexical subsetting, which looks at local acoustic features to eliminate words that the mapper would not accept. In either case, the lexical subset is intersected with the set of words in the predicted lexical subcategory to form the entries in the word set. (The word set is not created if the intersection is empty.) The word set is then assigned a priority and added to the list of word sets for use in the word task. The word set priority reflects the expected rating of terminal phrases constructed from words in the set. A single rating is computed for the entire set of words. If the system is mapping all at once, the priority of the word set is strongly influenced by the best mapper score for a word in the set. Otherwise, the priority is affected by the estimated false alarm likelihoods for words in the set, so that, other things being equal, the system will try words in an order that is expected to minimize false alarms. The priority of the word task is the highest priority of any word set.

a. GETTING A WORD AND CREATING A TERMINAL PHRASE

When it is performed, the word task begins by selecting a word from the highest priority word set. The selected word is the one with the highest mapper score, if the system is mapping all at once, or the one with the lowest false alarm likelihood, if the system is mapping one at a time. If there are no other words in the set, the set is deleted. Otherwise, the priority for the set is revised. If the system is not using the map-all strategy, the chosen word is now tested by the

mapper. If it is rejected, the word task goes directly to its final stage. In that stage, the priority for further word tests is compared to the priority for other system tasks. If word testing is still the highest priority, the word task is directly reexecuted. Otherwise, it returns control to the top-level Executive procedure.

If it is assumed that the word has been accepted by the mapper, the next operation is to create a terminal phrase (see Figure III-3). This operation begins by checking if a terminal phrase for the same word* in the same input location has already been created. For instance, the word might have been accepted as the result of a prediction from the opposite direction (right-to-left instead of left-to-right, say), or it might have been found following a prediction with a slightly different fixed time. If such a terminal phrase exists, the word task does not create a duplicate, but instead, it simply goes to its final stage.

If there is a terminal phrase for the same word and place, quit.

If there is a phrase for the same word at a different place, use the previous attributes and factors rather than recomputing them.

Otherwise, call the category procedure.

Construct the phrase record.

Distribute it to consumers.

Figure III-3. CREATE TERMINAL PHRASE

* In this discussion, a 'word' is a lexical entry, so if there is a word in category X that happens to have the same spelling or pronunciation as another word in category Y, they are still different words.

If there is a phrase already created for the word in a different location in the input, the attributes and language factors for that phrase can be reused rather than recalculated. For example, after a phrase for "it" is created, other "it" phrases use the same array of attributes, including the semantic network representation and the list of possible discourse referents. The shared language factors are combined with the particular mapper scores to produce scores for the different "it" terminal phrases.*

If the word has not been used for a previous terminal phrase, the category procedure is called to compute attributes and factors for it. If the category procedure does not reject the word, and the resulting phrase score is above a certain threshold, the terminal phrase record is constructed. The record holds the word's lexical entry, the times given by the mapper, the phrase score, and other information. The phrase is now ready for distribution to consumers.

b. DISTRIBUTING A PHRASE TO CONSUMERS

The procedure for distributing a phrase to consumers is the same for terminal and nonterminal phrases (see Figure III-4). It is called from the procedure that creates terminal phrases and from the procedure that adds a constituent to a consumer to create a complete

* Acoustic attributes and factors are not shared. In the current system, the only attributes that depend on acoustic results are the phrase times, and the only acoustic factors are from mapping and phrase mapping. The acoustic attributes and factors are treated specially by the system; there is not a general mechanism for dealing with them.

If it is a root-category phrase, give it to the RESPONSEFN.
For each prediction fulfilled by the phrase,
 Record the phrase as an instance of the prediction, and
 Add the phrase to each consumer of the prediction.

Figure III-4. DISTRIBUTE A PHRASE TO CONSUMERS

nonterminal phrase. If the category of the new phrase is the root category of the language, the phrase is passed to the language RESPONSEFN. This function saves the phrase for possible use in question answering and adjusts the Executive stopping parameters. If the category of the new phrase is not the root category, all predictions for the category are checked to see if the new phrase satisfies their time constraints. For example, if the phrase is an NP starting at location 35 and ending at location 55, it satisfies an NP prediction with left time 35 and right unfixed, but it does not satisfy an NP prediction with left unfixed and right at 180.

The actual algorithm for checking times takes two times as its input and decides whether or not they are compatible. For a phrase to satisfy a prediction, the left phrase time must be compatible with the left prediction time, and similarly for the right times. If a time is unfixed, it is compatible with any other time. Two fixed times are compatible if the gap between them is not too large. In the simulation experiments described in the next chapter, the allowed gap

was given by a parameter -- the standard size was 0.05 seconds, but this was varied in one of the experiments. With real rather than simulated acoustic processing, syllable boundary information is used in the time check -- the gap between two times must not contain an entire syllable. This test eliminates the obviously bad cases and leaves the more difficult ones to be handled by phrase mapping. Phrase mapping looks at a pair of words that have been accepted individually to see if they are acceptable as a sequence. Phrase mapping is done in the add-constituent operation when phrases are put together to form larger phrases (by a procedure discussed below).

For each prediction that the new phrase fulfills, the phrase is added to the prediction's instances list and then given to the prediction's consumers. The instances list is maintained so that consumers arriving later can make use of previously constructed instances. The operation of giving the phrase to a consumer is the source of nonterminal phrases that have one or more constituents.

2. ADDING A CONSTITUENT TO A CONSUMER

The add-constituent procedure (Figure III-5) performs preliminary tests to ensure that the phrase and its consumer are compatible with respect to times, phrase mapping, and lookahead. The time checks in the distribute-phrase procedure described above ensure that the phrase satisfies the times of the consumer's prediction; the time checks for the add-constituent procedure are more detailed and take

If the preliminary tests fail, quit.

Try to create a complete phrase, and distribute it if successful.

Try to create an incomplete phrase, and if successful then
assign its rating and schedule the predict task.

Figure III-5. ADD CONSTITUENT TO CONSUMER

into account other constituents of the consumer. If the preliminary tests succeed, the procedure goes on to try creating both complete and incomplete phrases.

The add-constituent procedure is called from two locations. It is called from the procedure that distributes new phrases, and it is called from the predict task when a new consumer is added for a prediction that has been previously fulfilled. Because of multiple consumers and multiple instances, a particular phrase can be added to many different consumers, and a single consumer can receive many different constituents. This multiple use is possible since constituents are not modified by their context, and consumers are copied before they are combined with a constituent.

a. PRELIMINARY TESTS

The add-constituent procedure begins with a series of tests to block certain bad constituent-consumer combinations. The first tests concern the time constraints on the new constituent imposed by the

time specifications of the consumer and its old constituents (see Figure III-6). The tests to the left of the new constituent will be described; similar tests are performed to the right. Let the index number for the constituent category arc be I . (This section assumes familiarity with the internal representation of the language definition. See Sections D.1 and D.2 of Chapter II.) If the new constituent has a left neighbor constituent in the consumer, the neighbor can be found by scanning through the consumer's constituent-array, starting at position $I-1$ and going down to position 1 looking for the first nonNIL entry (recall that entries are ordered from left to right in increasing positions of the array). If there is a neighbor, the following tests are made to ensure that it is time compatible with the new constituent. As an initial check, the left neighbor must really be somewhat to the left of the new constituent. If neither of the left or right times of the new constituent is to the right of the corresponding time of the left neighbor, the preliminary tests fail.

The next test depends on the structure graph relation between the new constituent and its left neighbor. If the consumer's structure graph indicates that the two phrases must be immediately adjacent (i.e., the right point of the left arc is the left point of the right arc), the rightmost word in the left phrase and the leftmost word in the right phrase are passed to the phrase mapping procedure. The job of the phrase mapper is to deal with coarticulation effects at word junctions, and if it rejects the pair of words, the add-constituent

If find a neighboring constituent to the left,

(1) quit if the neighbor is not to the left in the input,

(2) if the neighbor must be adjacent,
phrase map and quit if the test fails,

else if the neighbor optionally can be adjacent,
phrase map and block the NIL arc if the test fails.

Otherwise, if the consumer phrase left time is fixed and
the prediction left time is unfixed, then

if the new constituent must be leftmost,
check the left times and quit if the test fails

else if the new constituent can be leftmost,
check the left times and
block the NIL arc if the test fails.

Do the same tests to the right of the new constituent.

Figure III-6. PART 1 OF PRELIMINARY ADD-CONSTITUENT TESTS

procedure terminates without adding the phrase to the consumer. Alternatively, the structure graph for the consumer may indicate that the phrases do not have to be adjacent, but that they can optionally be adjacent if an unblocked NIL arc is used. In this case, phrase mapping is performed, and if it fails, the NIL arc is blocked to record that the phrases cannot in fact be adjacent. (The arc is marked as blocked by turning on the appropriate bit in a copy of the consumer's blocked-arc bit table.) The last alternative is that the structure graph does not allow the neighbors to be immediately adjacent. No phrase mapping is done in this case.

If there is not a left neighbor for the new constituent, but the consumer's left time is fixed, a time check may be made like the time checks to see if a phrase satisfies a prediction. However, if the consumer and the constituent were brought together by a prediction with a fixed left time, a time check is not necessary here since it would duplicate the check made when the phrase was added as an instance for the prediction. In case the prediction's left time is not fixed, but the consumer's left time is (which can happen if the consumer is a root category phrase), and the new phrase can be the leftmost constituent, a time check is made with the constituent's left time and the consumer's left time. If the test fails and the constituent must be leftmost, the add-constituent procedure terminates. If it can be leftmost by the use of an unblocked NIL arc and the time test fails, the NIL arc is blocked. These tests, and similar ones regarding the right side of the new constituent, ensure the acceptability of the times and word junctions between the new constituent and its consumer.

The second group of preliminary add-constituent tests look at the lexical subsets adjacent to the new constituent to block various arcs in the consumer (see Figure III-7). The lexical subsets are determined by acoustic tests and indicate the words that may be found in the utterance at the specified location and direction. For example, if the constituent starts at location 70 and ends at 105, the subset to its left will contain words that can end around 70, and the subset to its right will contain words that can start around 105. Each

For each arc coming in to the left of the new constituent,
block the arc if it is inconsistent with the lookahead.

If all arcs to left are blocked, quit.

Do similar tests to the right of the new constituent.

Block any arcs that can no longer be in a complete path.

Figure III-7. PART 2 OF PRELIMINARY ADD-CONSTITUENT TESTS

category in the language has a precomputed list of possible leftmost and rightmost terminal phrase categories. These lists are used to block arcs that are inconsistent with the lookahead provided by the lexical subsets.

The details of the tests to the right of the new constituent are given below; similar tests are also made to the left. If the arc for the constituent ends at the rightmost point of the graph, no lookahead tests are made at this point in the add-constituent operation (but more lookahead will be done at a later point in the operation if a complete phrase can be constructed). Otherwise, the arcs are checked that start directly to the right of the arc for the new constituent. If a category arc is blocked or filled, it does not need to be tested with respect to the lexical subsetting lookahead. Each unblocked, unfilled category arc is tested by intersecting the set of possible leftmost terminal categories for the arc with the lookahead set of categories for the lexical subset on the right of the constituent. If the intersection is empty, the arc is blocked. Next, each unblocked

NIL arc is checked that starts directly to the right of the constituent arc. If it leads to the rightmost point in the graph or if it leads to a filled category arc or to an unblocked, unfilled one whose leftmost terminal categories have a nonempty intersection with the lookahead categories, the NIL arc remains unblocked. Otherwise, it is blocked. If these tests leave no arcs unblocked directly to the right of the constituent arc, the add-constituent procedure terminates.

After the lookahead tests are completed to the left and right of the constituent, all the remaining unblocked arcs in the consumer are checked to make sure they can actually participate in a complete path through the structure graph. Arcs are blocked that are mutually exclusive with the arc for the new constituent. Any arc that cannot be in at least one complete path is also blocked by the following operation, which is done in two passes. The first pass goes through the consumer's array of arcs in increasing order so that all the left neighbors of an arc are considered before it is. If an arc does not start at the leftmost point in the graph and all of the arcs coming in to its left point are blocked, the arc is marked as blocked. The second pass goes through the arc-array in decreasing order. If an arc does not end at the rightmost point in the graph and if the arcs going out from its right are blocked, the arc is marked as blocked. To illustrate, assume that A has been added to an empty consumer with structure A {B C | D}, and lookahead to the right of A caused B to be blocked but left D unblocked. Arc C is marked as blocked during the first pass of this test because the only arc coming in to its left (arc B) is blocked.

The preliminary tests in the add-constituent procedure stop the procedure from trying to add a constituent to a consumer that is incompatible with respect to times, phrase mapping, or lookahead. Even if the constituent and the consumer are actually compatible, the tests still provide useful information by blocking arcs in the structure graph to reduce the number of possibilities that must be considered in later operations.

b. CREATE COMPLETE NONTERMINAL PHRASE

The procedure for constructing a complete nonterminal phrase (Figure III-8) starts with a test for a complete, filled path through the structure graph. If filled paths do not exist both from the right of the new constituent to the rightmost point in the graph and from its left to the leftmost point, the complete-phrase procedure terminates. The next test ensures that the new complete phrase will be compatible with its consumer context with respect to time constraints and lookahead. The details of this test are discussed below. If the test fails, construction of the new phrase is suspended pending the arrival of a new, compatible consumer. Assuming the test succeeds, the procedure checks whether it has already constructed a complete phrase using the same rule and constituents. If so, it terminates. The same phrase can be arrived at in different ways when the system uses control strategies that do not fix the order of acquisition of constituents. For example, with island driving, a phrase could be built both from the

Check for a complete, filled path through the structure graph,
and quit if there is none.

Check consumers regarding lookahead next to the new constituent,
and suspend construction if all consumers are blocked.

Check if the same phrase already exists, and quit if it does.

Check if a phrase with the same rule and equivalent constituents
exists already:

if it does, use the previous attributes and factors
rather than recomputing them;

Otherwise, call the rule procedure, and

if it rejects the phrase or gives it a subthreshold
score, quit.

Create a phrase record.

Distribute the new phrase to consumers.

Figure III-8. COMPLETE-PHRASE PROCEDURE

left and from the right. This test is necessary, therefore, to make
sure that duplicate phrases are not constructed.

Next, the procedure looks for a previously constructed
complete phrase for the same rule and 'equivalent' constituents. Two
terminal phrases are equivalent if they have the same lexical entry.
Two nonterminal phrases are equivalent if they were constructed by the
same rule and their constituents are equivalent. If two phrases are
equivalent in this way, they will have the same values for their
attributes and factors. Thus, if a phrase is found with the same rule
and equivalent constituents, its attributes and factors can be reused

rather than recalculated. Recall that a similar test was made for terminal phrases, and, as a result, equivalent terminal phrases share the same attribute array. The current tests ensure that nonterminal, equivalent phrases also share the same attribute array. This sharing saves storage and processing, and it also provides a simple test for equivalence -- two phrases are equivalent if their attribute arrays are the same. The search for a phrase with equivalent constituents is made more efficient by only considering one phrase from each class of equivalent phrases. If the new phrase is equivalent to a previous one, the old attribute array is used in the new phrase, and the old factors are combined with the new constituent scores and phrase mapping scores to produce a score for the new phrase. If no equivalent phrase exists, the rule procedure is executed to produce values for the attributes and factors, and the results are saved for future equivalence tests. If the rule procedure rejects the new phrase, or the phrase score is below a certain threshold, the complete-phrase procedure terminates. Otherwise, a record is constructed holding a pointer to the rule, the constituent list, the array of attributes, the score, and other information about the phrase. The newly made phrase is then distributed to consumers by the procedure described earlier.

A major step in the complete-phrase procedure is the consumer-lookahead test. This test is similar to the lookahead tests performed in the preliminary add-constituent stage, but it considers lookahead with respect to the consumer context of the new phrase being

constructed rather than within the consumer that is getting a new constituent. The purpose of this check is to prevent construction of complete phrases that cannot be used in the existing context of consumers. Building a complete phrase can require expensive semantic and discourse operations, so the system tries to discover an incompatibility before the phrase is constructed.

To illustrate the types of incompatibilities tested for by the consumer-lookahead check, assume that the new complete phrase A that is being constructed has a single consumer C. If the structure graph for C requires a phrase of category B to the right of A, but the set of categories that can occur as leftmost terminal phrases in a category B phrase do not intersect the lookahead set of categories for words immediately to the right of A, the phrase A is not compatible with the consumer C. Alternatively, C might already have a B phrase whose left-time boundary did not fit the right-time boundary of A. This case also causes the consumer-lookahead check to reject the consumer C for the phrase A. If all the consumers for A are rejected, the construction of A is suspended (without executing its rule procedure) until a new consumer arrives that is compatible with it.

The consumer-lookahead test looks both to the left and to the right of the new phrase, but the tests are similar, so only the right side tests are described in detail. The first step is to locate the arc in the consumer's structure graph that the new phrase would fill. (Remember that the new phrase we are talking about is the phrase

being constructed by the complete-phrase procedure, and the consumer now being discussed is a potential consumer of this new complete phrase.) The procedure looks to the right of the arc for either: (1) an unfilled, unblocked category arc that is not eliminated by lookahead, (2) a filled category arc with a constituent whose left time is compatible with the right time of the new phrase, or (3) an unblocked NIL arc that leads to a category arc satisfying case (1) or case (2).^{*} In any of the three cases, the phrase and the consumer are compatible to the right. If none of the cases succeeds, the consumer C may still be compatible with the phrase P if P can be C's rightmost constituent and the phrase resulting from adding P to C would be all right. P can be rightmost in C if its arc ends at the rightmost point in C's graph or if an unblocked NIL arc connects it to the rightmost point. If P can be rightmost, and the right boundary of C is fixed and compatible with the right boundary of P, P and C are compatible to the right. If P can be rightmost and the right boundary of C is unfixed, the consumer-lookahead procedure is called recursively to check consumers of C. For example, if C is of category X, the procedure checks consumers of C, which may have phrase structures such as W=X Y. In this particular case, the lookahead to the right of P would have to be compatible with a category Y phrase for the consumer-lookahead test to succeed.

^{*} We could have added phrase mapping in case (2) as a further test of consumer compatibility, but because phrase mapping is an expensive operation in our system, we decided that the extra sensitivity at this point would not justify the cost.

The consumer-lookahead procedure can thus go through an arbitrarily long path of intermediate consumers before getting to one that satisfies the lookahead requirements. Because of the structure of the parse net, these paths of consumers can form a network rather than a tree. To deal with this convergence, the procedure adds each prediction to a queue when the prediction's consumers are first reached. If the search returns to a prediction that is already on the queue, the consumers for that prediction are not checked again.

c. CREATE PARTIALLY FILLED NONTERMINAL PHRASE

If the preliminary add-constituent tests indicate that there will be at least one unblocked, unfilled category arc remaining after the constituent is added to the consumer, the procedure to create a partially filled nonterminal phrase is called (see Figure III-9). The procedure first checks if it has already made an incomplete phrase for the same combination of rule, constituents, and time specifications. If it has, it stops rather than creating a duplicate. Otherwise, it calls the rule procedure to compute the attribute and factor values.* If the rule rejects the phrase, or the score is subthreshold, the incomplete-phrase procedure quits. Next, a phrase record is constructed which has a pointer to the rule, the constituent list, the attribute

* We do not bother here to look for equivalent phrases as we did in the complete-phrase procedure because semantic translation and discourse processing are only done for complete phrases, and consequently the cost of recalculating the attributes and factors of an incomplete phrase is much less than the cost for a complete phrase.

Check for an unfilled, unblocked category arc; if none, quit.

Check if a phrase for the same rule, constituents, and time specifications already exists; if so, quit.

Call the rule procedure.

If it rejects the rule or gives a subthreshold score, quit.

Create a phrase record.

Calculate the phrase rating.

If the rating is above threshold, add the new phrase to the predict sets.

Figure III-9. CREATE AN INCOMPLETE PHRASE

array, the score, and other information. If the consumer that was used in this operation is a producer for some prediction (as will be the case unless it is a root-category consumer), the new phrase is also added as a producer for that prediction. The new phrase is then assigned a rating using the techniques described in Sections C.2 and D.5.c. If the rating is above a certain threshold, the phrase is added to the phrases scheduled to make predictions for missing constituents.

The incomplete phrases scheduled to make predictions are organized into 'predict sets'. Each predict set contains incomplete phrases that can make predictions at a particular time in a particular direction. To illustrate, if a phrase has a constituent ending at position 75 and needs to acquire a constituent to the right of that one, the phrase could be in a predict set for time equal to 75 and direction

equal to RIGHT. The priority of a predict set is initially set to the highest rating of any phrase in the set, but it can be modified in accordance with various strategies for focusing the system's attention. The priority of the predict task is the highest priority of any predict set.

A single incomplete phrase can be in several predict sets if it can make predictions at different locations and directions. For example, if the phrase structure calls for three constituents, A B C, and only the B phrase has been acquired, the phrase can be in a predict set for time equal to the left time of B and direction equal LEFT (to predict A), and also in a predict set for time equal to the right of B and direction equal RIGHT (to predict C).

The algorithm for adding an incomplete phrase to predict sets looks for all possible predictions that the phrase can make such that either the left or right time of the prediction is fixed.* For the purposes of this algorithm, a point in the structure graph is called 'open for predictions to the right' if there is an unfilled, unblocked category arc going out from the point to the right, or there is an unblocked NIL arc going out from the point that leads to an unfilled, unblocked category arc. First, if the left time of the phrase is fixed and the leftmost point in the graph is open for predictions to the

* We decided against having predictions with both times fixed. This choice sacrifices the ability to make word tests with both times fixed (such tests might succeed where tests with only one time fixed would fail), but it simplifies the word task and the predict task as well as reducing storage requirements.

right, the phrase is added to the predict set for time equal to the left of the phrase and direction, RIGHT. Then, for each acquired constituent, if the right point of the constituent arc is open for predictions to the right, the phrase is added to the predict set for time equal to the right of the constituent and direction, RIGHT. If the Executive is using a control strategy such as island driving that allows right-to-left predictions, a similar set of operations is performed to add the phrase to predict sets for direction equal LEFT.

3. PREDICT TASK

The predict task is divided into three main stages. In the first, a subnet of predictions and phrases is created for the highest priority predict set. The entire subnet is filled out at once: the predictions for the phrases in the predict set, the producer phrases for those predictions, the predictions for the new phrases, and so on. The second stage consists of going through the subnet assigning phrase ratings: whenever possible, all the consumers for a phrase are given ratings before the phrase rating is calculated. During the final stage, miscellaneous 'cleanup' operations are performed such as creating word sets for new predictions, revising priorities for old word sets, and adding old phrases to new consumers. The predict task is organized in this manner to reduce the processing time for rating phrases and setting priorities. The rating for a phrase is recalculated when its consumer context changes, and the rating calculation procedure uses the ratings

of consumers, so the predict task is designed to provide all of the consumers and finish giving them ratings before calculating the rating for a phrase.

a CREATE SUBNET

The create-subnet procedure, which performs the first stage of the predict task, is outlined in Figure III-10. While it creates the subnet, the procedure also sets up several queues for use in later stages. For the rating stage, it creates a queue, called PRQ, holding the predictions in the subnet that have up-to-date ratings for all of their consumers, and a second queue, called WPRQ, holding the other subnet predictions. For the cleanup stage, it creates LEXQ with the new predictions that need to have word sets made for them, LEXQ2 with old predictions that need to have the priorities for their word sets revised, INSTLIST holding new consumers for old predictions that have nonempty instances-lists, and OTHERINSTLIST holding consumers that provide a new context for phrases suspended in the complete-phrase procedure because of consumer-lookahead failure.

The create-subnet stage of the predict task begins by removing the highest priority entry from the list of predict sets. This entry will determine the predictions to be made in the rest of the task. The time and direction from the selected predict set are passed to the lexical subsetting procedure to retrieve the lookahead set of terminal categories. (Often, the subset will have already been computed for

```

Select the best predict set.

Get the lookahead information.

For each phrase P in the predict set
    For each prediction PR made by P
        If PR is newly created,
            Add it to PRQ and fill out its subnet

        Otherwise
            If it is not on PRQ or WPRQ, add it to PRQ

            Traverse its subnet

            If it has instances,
                Add P as consumer for it to INSTLIST.

```

Figure III-10. CREATE SUBNET PROCEDURE

lookahead during an earlier operation, so the stored results can simply be reused.) This information is used to block the creation of predictions and phrases that are incompatible with the lookahead. Next, the subnet is filled out for each phrase in the predict set.* As an

 * We do all of the phrases in the predict set at once rather than just doing the highest priority phrase. This is because we want to decrease the recalculation of priorities that is caused by changes in consumer contexts. This design decision was made before rules in our system could have alternatives and options. At that time, there were several rules had similar constituent structure possibilities (for example, five S rules with initial NPs), so it would often happen that several phrases with about the same priority would be waiting to make the same prediction in the same location. By creating the subnet for all of them at once, we can calculate ratings a single time rather than recalculating them each time one of the phrases makes its prediction which causes a change in the consumer context. However, with alternatives and options, predict sets are less likely to have several phrases that want to make the same prediction, so perhaps this design choice should be reconsidered in view of the fact that it does entail the risk of making unnecessary (i.e., not top-priority) predictions.

Illustration of this process, assume the predict set time is 55 and the direction is RIGHT. Then, for each predict set phrase P, the procedure finds the structure graph point at the right of the rightmost constituent that has a right time of 55, if such a constituent exists. Otherwise, P must have its left time fixed at 55, and the leftmost point in the graph is used in the following operation. For each unblocked, unfilled category arc A leading out of the selected point or connected to it by an unblocked NIL arc, find or create a prediction PR for the category of arc A, left time fixed at 55, and right time unfixed. If PR was newly created for P, the subnet below PR is filled out by a procedure described below, and PR is added to PRQ to record that all of its consumers have up-to-date ratings. Alternatively, if PR existed previously, then (1) it is put on PRQ unless it is already on a queue, (2) its subnet is traversed by a procedure described below, and (3) if PR has instances, P as a consumer for PR is added to INSTLIST.

The procedure used to fill in the subnet below a prediction (Figure III-11) begins by checking if the prediction is for a category with lexical entries. If it is, the prediction is added to LEXQ for further processing during the cleanup stage. The next step is to put a mark on the prediction so that if a later consumer arrives during this stage of the predict task, the subnet will not be reprocessed. The mark will be removed before the predict task is over. Then, in the case of a left-to-right prediction, for each rule that can produce a phrase of the predicted category, if the possible leftmost

To fill out the subnet for prediction PR with new consumer P:

 If the category of PR has lexical entries, add PR to LEXQ.

 Mark PR.

 For each of the category rules

 Create an empty phrase P' as a producer for PR

 For each prediction PR' made by P'

 Add PR' to WPRQ and, if necessary,
 remove it from PRQ.

 If PR' is newly created, fill out its subnet

 Else traverse its subnet, and

 If PR' has instances,
 add P' (as a consumer for PR') to INSTLIST.

Figure III-11. FILL-OUT-SUBNET PROCEDURE

terminal categories for the rule intersect the lookahead categories, an empty phrase P' is created for the rule with the same times as the prediction. The phrase is connected to the prediction as a producer, and then predictions are made in the following manner for the possible leftmost constituents of the phrase. For each category arc that can be leftmost and is okay with respect to the lookahead, a prediction PR' is found or created, and the phrase is added as a consumer for it. The prediction is added to WPRQ and removed from PRQ if it was there. This operation records the fact that the prediction now has at least one consumer whose rating is not set. If PR' was created by this operation, the subnet below it is filled in by calling this procedure recursively.

Otherwise, the subnet below the prediction is traversed according to the procedure described in the next paragraph. If it has instances, P' as a consumer for it is added to INSTLIST. Similar operations are performed to fill in right-to-left predictions.

The procedure for traversing the subnet below a preexisting prediction (Figure III-12) starts by checking for suspended construct-complete-phrase operations. If there are any, the consumer that led to the current traversal is added to OTHERINSTLIST for special processing during the cleanup stage. The procedure then checks to see if the prediction is marked as already traversed. If so, the procedure returns. If not, it marks the prediction and continues. The mark is checked after the (possible) addition to INSTLIST rather than before in order to take care of cases in which more than one consumer changes for an old prediction. If the prediction has word sets, it is added to LEXQ2 so that the word set priorities will be revised to reflect their new consumer context. Finally, for each producer phrase P for this prediction, each of the predictions made by P is put on WPRQ if it is not already there and traversed by calling this procedure recursively.

At the completion of the first stage of the predict task, the subnet of predictions and phrases has been created for all the phrases in the highest priority predict set. The queues PRQ and WPRQ have been created for the rating stage, the queues LEXQ and LEXQ2 are ready for use in creating or revising word sets, and INSTLIST and

To traverse the subnet for a prediction PR with a new consumer P:

If PR has any suspended construct-complete-phrase operations,
add P (as a consumer for PR) to OTHERINSTLIST.

If PR is marked, quit.

Mark PR.

If PR has word sets, add it to LEXQ2.

For each prediction PR' by a producer for PR

Traverse subnet of PR' and put PR' on WPRQ.

Figure III-12. TRAVERSE SUBNET PROCEDURE

OTHERINSTLIST hold consumers that may lead to the creation of new phrases.

To illustrate the create-subnet procedure, assume that the highest priority predict set has a time equal to the right boundary of the utterance, a direction equal to LEFT, and a single phrase P1 which has a structure declaration S=BE NP (see Figure III-13). The only prediction to be made by P1 is for an NP ending at the right of the utterance. Assume that this prediction has not been created by a previous predict task (if it had been, the traverse-subnet procedure would be called rather than the fill-out-subnet procedure). The prediction PR1 for the NP is created, added to PRQ, and passed to the fill-out-subnet procedure. The NP category has lexical entries such as "it", so PR1 is added to LEXQ for further processing in the cleanup stage. The prediction is then marked so that it will not be

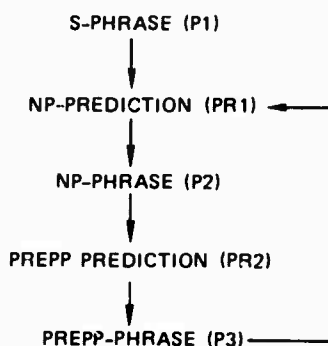


FIGURE III-13 NP-PREPP PARSE NET LOOP

reprocessed. An empty NP phrase P2 is created as a producer for PR1, and it makes a prediction PR2 for a PREPP at the end of the utterance and adds PR2 to WPRQ. (Other predictions would be made by P2, but for simplicity we only consider PR2.) Again assuming PR2 did not exist before, it is passed recursively to the fill-out-subnet procedure. The PREPP category does not have lexical entries, so PR2 is not added to LEXQ. PR2 is marked, and a producer P3 is created for it. P3 makes a prediction for a noun phrase at the end of the utterance, but that is prediction PR1, which was already created. Therefore, PR1 is moved to WPRQ from PRQ to record that it has a consumer without an up-to-date rating. Then, PR1 is passed to the traverse subnet procedure which quits after noticing that PR1 is already marked. Assuming for simplicity that no other predictions or phrases are formed, the first stage ends with LEXQ holding PR1, WPRQ holding PR1 and PR2, and the other queues empty.

b. ASSIGN RATINGS

The second stage of the predict task takes care of assigning ratings to phrases in the subnet. The algorithm for calculating the rating of a particular phrase is sketched in Section C.2 above and is described in detail in section D.5.c below. This section deals with the procedure that goes through the subnet visiting the phrases and calling the rating procedure for each one. In the simplest case, a single top-down pass is made through the subnet, but the operation can be more complicated if there are producer-consumer loops. A top-down pass is desirable because the calculation of ratings takes advantage of the ratings for the consumers for the phrase. The predictions placed on PRQ during the first stage have up-to-date ratings for their consumers, so the producers for those predictions are ready to have their ratings calculated. When a phrase gets its rating, each of its predictions is checked, and, if all of the prediction's consumers now have up-to-date ratings, the prediction is moved to PRQ from WPRQ. If there are no loops in the subnet, repeated removal of predictions from PRQ followed by processing of their producers in the above manner, will eventually provide ratings for all the subnet phrases in the desired top-down way. If there is a loop in the net, it is not possible to assign ratings in a strictly top-down manner since phrases in the loop are consumers for themselves. When there is a loop, the algorithm goes as top-down as possible and makes a second pass to recalculate ratings in certain cases. The following paragraphs describe the

algorithm in detail and illustrate it by continuing with the example from the previous section.

The first pass of the rating stage is an iterative operation that continues until both PRQ and WPRQ are empty (see Figure III-14). When both queues are empty at the start of an iteration, the second pass begins. If PRQ is not empty, the first prediction on it is removed and set aside for processing. If PRQ is empty, but WPRQ is not, the subnet has a loop that includes the WPRQ predictions. However, at least one of the WPRQ predictions must have one or more consumers that are not involved in the loop. If this were not the case, the predictions on WPRQ would not be reachable from the rest of the parse net, but they are in fact reachable from at least one of the phrases in the current predict set. A prediction on WPRQ with at least one rated consumer is removed and set aside for processing. The prediction is also marked as DONE so that it can be recognized when the first pass returns to it after completing the loop. The processing of the selected prediction is the same whether the prediction is from PRQ or WPRQ. Each producer phrase P for it is given a rating, and then for each prediction PR by P, (1) if PR is on WPRQ, then if all the consumers of PR now have up-to-date ratings, PR is moved to PRQ, else (2) PR is checked to see if it is marked DONE. If it is marked DONE and the rating just calculated for its 'looping' consumer is higher than the ratings of all of its other consumers, PR is added to PASS2Q for further processing in the second pass. At this point, control goes back to the start of the first pass instructions to check again whether PRQ and WPRQ are empty.

If PRQ and WPRQ are both empty, go to pass 2.

Pick a prediction from PRQ, or,

If PRQ is empty, pick one from WPRQ that has at
least one rated consumer.
Mark it DONE.

For each producer P of the selected prediction

Assign a rating to P.

For each prediction PR made by P

If PR is on WPRQ, then,
If all consumers of PR now have ratings,
Move PR to PRQ

Else if PR is marked DONE

If the rating for P is higher than the
rating of any other PR consumer
Then add PR to PASS2Q

Otherwise, there is an error.

Figure III-14. PASS 1 OF RATING ASSIGNMENT

The second pass is an iterative operation that continues until PASS2Q is empty (see Figure III-15). If PASS2Q is not empty, a prediction is removed from it and marked as DONE2 so that it will be recognized if the second pass returns to it. Each producer phrase for the prediction has its rating recalculated, and, if its rating changes, each prediction made by the phrase is added to PASS2Q if the prediction is not marked DONE2 and the other consumers for P have lower ratings than the rerated consumer. When the second pass terminates, all the phrases in the subnet have up-to-date priorities.

If PASS2Q is empty, quit.

Pick a prediction from PASS2Q.

Mark it as DONE2.

For each producer P of the selected prediction

Recalculate the rating for P.

If the rating changed,

For each prediction PR by P not marked DONE2

If P is now the highest rated consumer for PR,
Add PR to PASS2Q.

Figure III-15. PASS 2 OF RATING ASSIGNMENT

The second pass is basically a patch to take care of loops in the subnet. If there are no loops, the second pass is vacuous because no predictions are put on PASS2Q unless there is a loop. Even if there are loops, the second pass will only be lightly used if ratings tend to decrease as consumer paths get longer (which is plausible since longer paths represent more complex, and hence less likely, structures). This is because there is always a shorter alternative path to a looping path, and predictions are added to PASS2Q only if a looping path has a higher rating than any of the others. Activity in the second pass will also be slight if ratings do not change when they are recalculated in a context that is unchanged except for the ratings of one or more consumers (such stability will in fact be the case if the context-checking method is used for calculating ratings).

To illustrate this stage of the predict task, we continue the previous example (see Figure III-13). Recall that the first stage for the example ended with PRQ empty, and PR1 and PR2 on WPRQ. At the start of the second stage, PR1 is selected from WPRQ since it has a rated consumer (P1) and PR2 does not. PR1 is marked DONE, and its producer P2 has its rating assigned. The only prediction for P2 is PR2. PR2 is on WPRQ and now has all of its consumers rated, so it is moved to PRQ. Thus, the first iteration ends with PR2 on PRQ, P2 with a rating, and WPRQ empty. The second iteration starts by removing PR2 from PRQ and then assigning a rating to its producer P3. The only prediction by P3 is PR1, which is not on WPRQ but which is marked DONE. Assuming that the rating of P3 is higher than the rating for P1, PR1 is added to PASS2Q. The second iteration ends with PR1 on PASS2Q, P2 and P3 with ratings, and both PRQ and WPRQ empty. At this point, the second pass begins by removing PR1 from PASS2Q, marking it as DONE2, and recalculating the rating for P2. Assuming that the recalculation produces a higher rating for P2, its prediction PR2 is added to PASS2Q. Thus, the first iteration of pass two ends with PR1 marked DONE2, P2 rerated, and PR2 on PASS2Q. The second iteration begins by removing PR2 from PASS2Q, marking it DONE2, and recalculating the rating for P3. The prediction made by P3 is marked DONE2, so it is not processed further and the second pass ends. .

c. CLEANUP

The final stage of the predict task performs various cleanup operations. It erases marks such as DONE2, which may have been left by the previous stage. Predictions and phrases needing erasures are chained together through their records, so the erasing can be done quickly. LEXQ holds new predictions for categories with lexical entries. For each prediction on LEXQ, word sets are created in the manner described earlier (see Section D.1). LEXQ2 holds old predictions with word sets that now have a modified consumer context. Each such word set has its priority recalculated. INSTLIST contains new consumers for old predictions that have been previously satisfied. Each phrase on the prediction's instances-list is added to the new consumer by the add-constituent procedure. OTHERINSTLIST contains consumers providing a new context for old predictions with suspended construct-complete-phrase operations. The consumers on OTHERINSTLIST may be new consumers that were created during the first stage or old consumers that had an addition made to their consumer context during the first stage. The traverse-subnet procedure takes care of both cases. During the cleanup stage, for each suspended operation with a new or modified consumer P, the consumer-lookahead test is performed. If the test succeeds, the construction of the phrase goes ahead. Otherwise, it remains suspended.

After the cleanup stage, one cycle of the predict task is complete. However, there may still be other predict sets waiting to be processed. If there are, and the highest priority for them is higher

than any of the tasks currently scheduled, the predict task goes ahead for another cycle. Otherwise, it schedules itself at the priority of the best predict set and returns control to the top level Executive procedure.

d. DEAD PHRASES AND PREDICTIONS

During the predict task, certain phrases and predictions are marked as 'dead'. A prediction is dead if no more phrases can possibly be constructed to fulfill it. A phrase is defined to be dead if it has no more predictions to make and all the predictions it has made are dead.* If a phrase is dead, no tasks exist to supply constituents for it, and, thus, no task priorities depend on its rating. Consequently, ratings do not need to be (re)calculated for dead phrases. Typically, the rating of a phrase is recalculated whenever the consumer context of the phrase changes. The purpose of marking phrases as dead is to avoid unnecessary rating recalculations.

The marking of predictions and phrases as dead takes place during the first stage of the predict task. Whenever all the candidate words for a prediction are exhausted in the word task or one

* Intuitively, a phrase is dead if there is no longer a chance to form new completions of it. In some cases, a phrase may be dead in this intuitive sense but not according to our definition. For example, a phrase with structure A B C that has acquired constituent B and has predicted A and C is intuitively dead as soon as either prediction dies, but it is not dead by our definition until both predictions die. Our definition was chosen in spite of this defect because it is simpler to test (it does not depend on the structure) and in practice it is usually equivalent to the intuitive sense.

of the the prediction's producers is killed by the procedure described below, the prediction is given a special marker indicating that it should be checked to see if it is still alive. The predict task procedure for traversing the subnet below a preexisting prediction looks for this marker. If the marker is found, the procedure searches the producer subnet for a phrase that still can make more predictions or a prediction with candidate words remaining. This search takes care of recognizing dead predictions even if they are part of consumer-producer loops. If the search fails, the prediction is 'killed'; in other words, it is marked as dead, and for each consumer of the prediction, if all of the consumer's predictions are now dead and the consumer has made all of its predictions, the consumer is killed. A consumer is killed by marking it as dead and if it is a producer for a prediction PR, and PR does not have any remaining candidate words, then either PR is killed if all of its producers are now dead, or it is marked to indicate that it should be checked to see if it is still alive if a subsequent predict task encounters it. This method propagates markers up the parse net during the first stage of the predict task. During the second stage, ratings are not recalculated for phrases marked as dead.

Dead phrases cannot be discarded from the parse net during the first stage of the predict task because they may be needed as consumers during the cleanup stage. For example, if a phrase P makes a single prediction PR and PR is dead but has instances, then P will be marked dead during the first stage, but it will have the instances of PR

added to it during the last stage. It would be possible, of course, to keep a list of killed phrases and prune them at the end of the cleanup stage, but we have not implemented such a scheme.

4. MULTIWORD LEXICAL ENTRIES

Small words like "of", "are", "a", and "the" cause difficulties for a speech understanding system because they are often poorly articulated or reduced to a short duration remnant. In order to minimize false rejections for such words, the acoustic mapper must use very loose criteria in testing them. However, the loose criteria lead to a high false alarm rate. This creates a severe problem because the language definition often allows several small words to occur in a series. To reduce the bad effects of small words, we have introduced a mechanism allowing a series of small words to be processed acoustically as a single unit. With this approach, a series of words is combined to produce a larger 'multiword lexical entry', or 'multiword', which can be tested more reliably. Small words are tested individually only for contexts that do not put other small words beside them. A set of over 50 multiwords was used in the experimental tests of the system. They included phrases such as "is the", "of a", and "what are the".

This method does not eliminate the small-word problems, but preliminary results suggest that it does reduce them. Following the introduction of multiwords, the mapper criteria for small words were tightened to reduce their false alarm rate. Data collected on the

mapper performance (see the discussion in Chapter IV) included two cases in which small words were incorrectly rejected, perhaps because of this tightening. However, in both cases the missed words were part of a series of small words, and the multiword for the series matched well enough to be accepted in spite of the fact that one of its component words was rejected when mapped alone. Thus, the sentences could be understood correctly although one of the (small) words was incorrectly rejected. More tests are required to assess the effects of multiwords, but these initial results suggest that they may provide a means of raising the accuracy of acoustic processing.

The multiwords are treated as single units for acoustics, but not for linguistic processing. The language definition would be badly distorted by an attempt to incorporate multiwords directly. Instead, the Executive breaks multiwords into their individual parts, and multiword entries like "is a" are analyzed by the linguistic knowledge sources as two separate words although recognized by the mapper as one. When a multiword with individual words W_1, \dots, W_n is accepted from position P_1 to position P_2 in the input, the Executive creates $n-1$ new 'pseudo-positions', X_1, \dots, X_{n-1} , which are distinct from positions in the actual input. The multiword is then analyzed as a series of regular words: W_1 from P_1 to X_1 , W_2 from X_2 to X_3 , ..., and W_n from X_{n-1} to P_2 .*

* This method is similar to the Kay and Kaplan method of representing the input as a 'chart' (see Kay, 1967, 1973).

The pseudo-positions must be treated as special cases in some of the Executive algorithms. The time compatibility tests that are used by the distribute-phrase and add-constituent procedures look for pseudo-positions. If both times being tested are fixed and one is a pseudo-position, the other must be the same pseudo-position or the times are incompatible. The phrase mapping procedure accepts without testing two adjacent words from the same multiword phrase. However, if the words for phrase mapping are not from the same multiword, either word that is from a multiword is replaced by the entire multiword before phrase mapping takes place. For example, if called with the pair "the" and "ship" for phrase mapping and "the" comes from the multiword "is the", the phrase mapping is done using "is the ship". If the lexical subsetting procedure is called with a pseudo-position, it does not do any acoustic processing, but instead looks in a table of accepted multiwords to find the word from the multiword phrase at the given position. Thus, from the $W_1 \dots W_n$ example, the subset to the right of X_1 contains just W_2 .

Multiwords are standard sequences of words from the language, so there is a danger of wasted effort expended analyzing both a multiword and the individual words composing it. For instance, if the multiword phrase "of the" is accepted from position 35 to position 55, the mapper may also accept "of" from 35 to 45 and "the" from 45 to 55. The Executive has two ways of blocking series of small words that duplicate a multiword. First, the phrase mapping routine rejects pairs

that together form a multiword. Thus, a phrase whose rightmost word is "of" cannot be put to the left of a phrase whose leftmost word is "the" unless the "of the" comes from a single multiword accepted by the mapper. The second block on small words that duplicate a multiword occurs in the word task. If "of" is the only word accepted ending at position 45, "the" will be excluded from the candidate words starting at 45 because "of the" is one of the system multiwords. The exclusion is not permanent; it will stop if another word ending at 45 is later found and leads to a new execution of the predict task at position 45 and direction equal RIGHT. The exclusion is removed as part of the word set revisions during the cleanup phase of the predict task.

Recall that the Definition Compiler adds the multiwords to the lexicon so that they are available for the word task algorithms. Multiwords beginning with word W are added to all lexical subcategories containing W and are marked for use in left-to-right word sets. Similarly, multiwords ending with W are added also, and are marked for use in right-to-left word sets. The procedure that creates word sets eliminates multiwords that are marked for the wrong direction. The remaining multiwords are treated like standard words until they reach the procedure to create terminal phrases. This procedure recognizes the accepted word as a multiword, creates pseudo-positions for it, and then creates a terminal phrase for one of the standard words in the phrase. If the multiword lexical entry was marked for left-to-right use, the terminal phrase is constructed for the leftmost word in the multiword.

Otherwise, the terminal phrase is created for the rightmost word. The terminal phrase is distributed and added to consumers in the standard manner. The resulting incomplete phrases lead to predictions at the pseudo-position within the multiword. The standard parse net structure and task algorithms are used in analyzing the multiword, and the analysis benefits from the precise lookahead information available at the pseudo-positions.

In summary, multiword lexical entries appear to offer a way to improve acoustic accuracy and, by the methods outlined above, have been integrated into the system in a simple manner without requiring any changes in the language definition.

5. PRIORITY SETTING

Newell and Simon (1976) conclude their 1975 Turing Award Lecture by stating:

For all physical symbol systems, condemned as we are to serial search of the problem environment, the critical question is always: What to do next? (p.126)

In our system, this critical question is answered according to task priorities, thereby replacing the original question by another difficult one: How to set priorities? The approach we have taken is to base priorities on phrase ratings that combine results from acoustic mapping and language factors. The predict task priority is the highest

rating for an incomplete phrase waiting to make a prediction. The word-task priority is the highest rating for a terminal phrase from the current word candidates. The priorities are initially derived from ratings, but they can be modified according to various strategies such as focus by inhibition.

a. FACTORS

Phrase ratings are derived from phrase scores, and phrase scores are in turn derived from language factors and mapper factors. The language factors are defined in lexical-category procedures and rule procedures. Many of the language factors are Boolean. Such factors block the construction of a phrase if the phrase would have certain undesirable properties. If the properties are not present, the Boolean factor allows the phrase to be constructed and does not affect the phrase score. The remaining language factors are non-Boolean. They raise or lower the phrase score according to the results of various tests and are used to 'tune' the language definition (as discussed in Robinson, 1975b). For example, if questions are expected to be the predominant form of input to the system, non-Boolean factors can be added to raise scores for questions relative to other kinds of sentences. Such factors would bias the system toward looking for ways to interpret utterances as questions.

Boolean language factors are implemented as calls on the F.REJECT procedure, which causes the construction of a phrase to be

immediately terminated. The non-Boolean factors are implemented as variables assigned integer values in the range 0 to 100 or pairs of integers, <WEIGHT TOTAL>, such that TOTAL divided by WEIGHT is between 0 and 100. A factor value that is a single integer J is equivalent to the pair <1 J>. The factor WEIGHTs reflect relative importance; a factor with WEIGHT equal 2 has twice the effect on the phrase score as a factor with WEIGHT equal 1.

Mapper factors are derived from mapper scores and thus are based on acoustic, phonetic, and phonological judgments. Mapper factors for individual words and multiword lexical entries directly contribute to the scores of terminal phrases. Similarly, phrase mapping factors contribute to the scores of nonterminal phrases. A nonterminal phrase with N constituents has N-1 phrase junctures and N-1 phrase mapping factors. For both word mapping and phrase mapping, the mapper score is an integer ranging from a top value of 100 down to a threshold of 45.

Mapper scores are converted to mapper factor values according to the estimated likelihood that the word is a false alarm. Each word and multiword has a false alarm rating in the range 0 to 100, indicating the relative likelihood of producing a false alarm compared to the words in the vocabulary that are considered the most likely to produce false alarms. The most likely words get a false alarm rating of 100, words half as likely get a rating of 50, and so on. The ratings are rough estimates originally provided by the implementors of the

mapper and revised upward in some cases according to experimental results.

The false alarm rating provides a confidence measure for mapper results, and the system gives a greater weight to mapper factors for words with low false alarm ratings. The factor WEIGHT is 1 if the rating is above 90, 2 if it is between 81 and 90, 3 if it is between 71 and 80, and similarly up to 10 for ratings between 0 and 10. Also, the range of mapper factor values is reduced for words with high false alarm ratings. This reduction causes the system to adopt a casual attitude toward very high mapper scores on small words like "a". The mapper scores are converted by a linear transformation with a slope that decreases as the false alarm rating of the word increases. For a minimum false alarm rating of 0, the result is 27 plus two-thirds of the mapper score; scores in the range 45 to 100 are converted to results in the range 57 to 94. For a maximum rating of 100, the result is 44 plus one-third of the mapper score; scores in the range 45 to 100 produce results in the range 57 to 77. After the weight has been calculated and the score transformed, the mapper factor is constructed. It is a pair: the WEIGHT as determined by the false alarm rating of the word and the TOTAL equal to the product of the WEIGHT and the transformed score. Ideally, this ad hoc method would be replaced by a scoring technique based on greater information about the performance characteristics of the acoustic processor (perhaps like the probabilistic scoring described by Klovstad in Woods et al., 1975a, pp.33-39).

b. PHRASE SCORES

The score for a phrase combines factors to form a WEIGHT and TOTAL pair that can itself be used like a factor value. For a terminal phrase, the score combines language factors from the lexical-category procedure and the mapper factor for the word. For a nonterminal phrase, the score combines factors from the rule procedure, phrase-mapping factors, and constituent scores. In most cases, the score is simply a vector sum of the factors: the WEIGHT is the sum of the factor WEIGHTs, and the TOTAL is the sum of the factor TOTALs.

To explain the cases in which the score is not just a vector sum of the factors, we define the 'Q' of a score or factor to be the quotient of the TOTAL divided by the WEIGHT. The Q is important because it provides a 'quality' dimension for comparing factors and scores, and the Q of scores for root-category phrases is used in determining phrase ratings. In case the phrase score is a vector sum of the factors, the Q of the score is a weighted arithmetic mean of the Qs of the factors. Thus, the Q of the score is insensitive to the total number of factors; five factors with average Qs give a result that differs from that for ten factors with average Qs only in weight, not in Q. This insensitivity is desirable, because the factors in our system are not like independent probabilities and combining them by a technique that was sensitive to the total number of factors would be inappropriate. However, there would be a danger with using the arithmetic mean that a very low factor would fail to pull the score down

as much as it should. A possible solution would be to use a geometric mean instead (the geometric mean of N numbers is the N th root of their product). We have taken a different course and treat low factors as special cases; if the Q of the factor is below a certain threshold by X percent, the TOTAL for the score is reduced by X percent. The threshold in the current implementation is 50, so a Q of 40 causes the TOTAL to be reduced by 20%. (Details of the algorithm for combining factors into scores were given in Section II of Walker et al., 1975).

c. PHRASE RATINGS

The rating for a phrase P is a certain constant times the Q of the score for a root-category phrase that could be constructed using P . (The constant is chosen to spread the phrase ratings over a wide range of integers.) The rating for a root-category phrase comes directly from the Q of its score. The rating for a nonroot phrase is derived by operations that look at the consumer context of the phrase. As discussed in the preceding overview, we have experimented with two techniques for calculating phrase ratings. The two methods for assigning nonroot phrase ratings are 'context checking', which entails the execution of consumer rule-procedures to gather factor information, and 'merging', which does not. Both methods take advantage of the fact that the system is designed to assign ratings to the consumers of a phrase P before assigning a rating to P itself. Define the 'rating-score' of the consumer to be the WEIGHT and TOTAL pair forming the score

used in calculating the consumer rating. Also, let the 'merged rating-score' of P and a consumer C be the vector sum of the score of P and the rating-score of C. The rating of P with respect to the consumer C is determined from the Q of the merged rating-score of P and C. The rating of P, as calculated by merging, is the highest rating of P with respect to any of its consumers. (Recall that there is always at least one consumer for a nonroot phrase.)

The merging method is fast and simple. Moreover, it reflects both the quality of the phrase and the quality of the consumers of the phrase. However, it does not indicate whether the phrase is really compatible with its consumers. The phrase may have attributes that will cause any complete phrase built from it to be rejected by the consumers. There is nothing to be gained from working on such a phrase, but the merging method will not give it a low rating if its score is good and the consumers' ratings are good. The context-checking method is designed to avoid this defect.

To start with a simple example, assume that the phrase P has a single consumer C, and C is a root-category phrase. The context-checking method assigns a rating to P by creating a 'virtual' phrase C' from C and P. Virtual phrases are only created during context checking and are deleted immediately afterward with all storage reclaimed. By assumption, C' is a root category phrase, so the score of C' is used to determine the rating of P.

If C is not root category, but has a single consumer D which is, the context-checking method adds C' to D to form another virtual phrase D', and the score of D' determines the rating for P. In general, any path leading from P to a root-category consumer can be used to establish a rating. The phrase rating could be defined as the maximum rating with respect to any consumer path leading to a root-category phrase. However, there may be many such paths, and they would all have to be tried in order to guarantee finding the best one. The cost of such an exhaustive exploration of the consumer context could overshadow the potential benefits, so the system instead uses heuristic methods to find a near-optimal path without necessarily considering all of the paths.

The details of the context-checking algorithm for setting phrase ratings are as follows. If the phrase P to be given a rating is a root-category phrase, its rating is calculated directly from its score. Otherwise, a lower bound for the rating is initialized and the consumers of P are scheduled for the creation of virtual phrases. The initial lower bound is zero if P is being rated for the first time. In case P is having its rating recalculated because of some addition to its consumer context, the lower bound is set to its prior rating. Each consumer C is scheduled at a priority equal to 98% of the rating determined by merging the score of P with the rating-score of C. The 98% is included as a 'laziness' factor to cause the process to stop sooner. (Note that the scheduling referred to here is internal to the rating algorithm; it is not part of the Executive task scheduling.)

After the lower bound is initialized and the consumers scheduled, the path growing begins. It continues until there are no more extensions scheduled or the highest priority is lower than the lower bound. At each step in the growing, the highest priority extension is performed. A scheduled extension consists of a consumer C and a (possibly empty) path of virtual phrases. Let X be the most recently added virtual phrase in the path or P if the path is empty. In either case, C is a consumer for X, and the attributes and score of X are available for use in constructing a new virtual phrase, C'. If C' is rejected by the rule procedure or if its score is subthreshold, this step in the path growing is terminated. Otherwise, if C' is root category, its score is used to update the lower bound. If C' is not root category, extensions are scheduled for each of its consumers at a priority of 98% of the rating determined by merging the consumer's rating-score and the score of C'. The consumer is not scheduled if its priority is less than the lower bound, or if it already appears twice in the path leading to C' (thus blocking paths from going around consumer-producer loops more than once).

The merging method and the context-checking method give similar results for phrases that have no conflicts with their consumer contexts. However, when a phrase has attributes that would cause completions of it to be rejected or downgraded by consumers, the context-checking method gives it a lower rating than the merging method. If the rule procedures are a major source of evaluative knowledge,

context-checking can produce more useful ratings and hence better priorities. However, merging is a faster method and can produce results that are as useful if the rule procedures provide only limited constraints. In our system, the rule procedures are in fact an important source of information, and experimental results show that the use of the context-checking method has good effects on performance.

One of the choices considered in the control strategy experiment (reported in Chapter IV, Section E) was whether or not to use context-checking in setting ratings. Some of our test systems used the merging method exclusively. Other test systems used a mix of the methods: context-checking for partially completed nonterminal phrases, and merging for empty ones. This mix is preferred to pure context-checking because empty phrases have so few attributes established that they are almost always compatible with their context. By mixing the methods, the costlier checking is only done in cases where it is more likely to help.

C. RELATION TO EXECUTIVE TASKS

Ratings are calculated at several points in the Executive tasks. When an incomplete phrase is created in the add-constituent operation, it is inserted in the parse net to establish its consumer context, and then its rating is calculated. If its rating is above a certain threshold, the phrase is added to the predict lists. Otherwise, the phrase will not be allowed to make any predictions unless its consumer context changes to raise its rating above the threshold.

During the second stage of the predict task, new phrases get initial ratings, and old phrases with modified consumer contexts get revised ratings. In the case of a revised rating, if the phrase had been kept out of predict sets previously, but now has an above threshold rating, it is added to the predict sets and becomes eligible to make predictions. Otherwise, if the phrase is a member of a predict set and was or is now the best phrase in the set, the priority of that set is updated to reflect the new phrase rating. These revisions ensure that a phrase that is given a low initial rating has a chance to get a higher rating later if its consumer context changes. The revisions also keep the priorities in step with the ratings.

Like predict sets, word sets have priorities that depend on ratings. These ratings are calculated as part of the cleanup stage of the predict task. They are initially calculated when the word set is created and are revised when the consumer context changes. The consumers that are used in setting a word set rating come from the prediction that caused the creation of the word set.

The details of setting word set ratings depend on whether the system is mapping all at once and also on the number of words in the set. When mapping all at once, the words in the set have mapper scores and corresponding mapper factors. The word set is assigned a score equal to the mapping factor with the highest Q. The rating of the word set is then determined by the merging method.

If the system is not doing mapping all at once, the word set rating is calculated either by merging (using a default score for the word set) or by context-checking. The context-checking makes virtual phrases using a default score and the default attributes of the word set lexical subcategory. The defaults are shared by all the words in the subcategory, so only one rating needs to be calculated for all the words in the word set. The savings resulting from this are the principal motivation for having lexical subcategories. The merging method is employed if the system is using a no-context-checking control strategy, or if there are few words in the word set (three or less, in the current implementation).

The use of merging when there are few words in the word set represents a trade-off between processing time for context checking and processing time for acoustic tests. If the context checks are performed and result in a low rating, the system may avoid doing unnecessary acoustic processing. If merging is used and results in an inflated rating, extra acoustic tests may be done. The larger the word set, the more likely it is that the cost of the context checking will be offset by savings in acoustic processing since the context checking is done only once for the entire set but the acoustic tests are repeated for each word.

6. ADJUSTING PRIORITIES AND FOCUS BY INHIBITION

Phrase ratings determine initial priorities, but priorities can be adjusted according to a variety of strategies. If no adjustments are made, the system will work in a 'best-first' manner strictly following the ratings in moving from one task to another. Looking at the series of tasks performed by such a best-first method, one often observes the system shifting its activity among competing possibilities at a high rate. Such observations led to an experiment in adjusting priorities in hopes of making the system 'focus attention' better. The method of adjusting priorities is called 'focus by inhibition' because it affects the system focus of attention by inhibiting work on certain alternatives.

To explain the motivation for focus by inhibition, assume, contrary to fact, that the system could find a word X in the input that was guaranteed not to be a false alarm. Having found X, the system would not want to waste time on any task that would try to replace it. In other words, the system should block word tasks that would produce words overlapping X and block predict tasks for phrases containing words that overlap X. In this way, the system would avoid wasting its effort on tasks that were in conflict with a word that was sure to be correct. In actual practice, the system is only relatively sure of words rather than being absolutely sure. Instead of being sure that X is correct, the system will have a certain confidence that X is correct. If it is highly confident in X, it will be reluctant to perform conflicting

tasks, but it will not absolutely refuse to perform them because X may be a false alarm. Focus by inhibition implements this reluctance as a reduction in priority of conflicting tasks. The priority reduction causes a system bias, but it is a bias that can be overcome -- the task can still become top priority in spite of having its priority reduced.

In the implementation of focus by inhibition, the focus is represented by a set of time-compatible words (or multiwords) from the input. Two words are time-compatible if they do not overlap or have a small enough overlap that they could still be in the same phrase. Words are time-incompatible if they are not time-compatible -- in other words, if they overlap so much that they cannot be in the same phrase. A phrase conflicts with focus if any word in the phrase is time-incompatible with some word that is in focus. The phrase priority is initialized to its rating and is lowered if the phrase conflicts with focus. The percentage amount by which the priority is lowered depends on the 'strength' of the focus word. The focus strength goes up according to the mapper score for the word and the ratings of the phrases putting the word in focus. The strength decreases according to the likelihood that the word is a false alarm.

At the start of the predict task, the top priority phrase P is selected from the top priority predict set. It is possible that P became top priority in spite of being in conflict with some focus word W. If so, W is removed from focus and any task inhibited because of conflict with W has its priority raised to its preconflict level. Next,

the predict task checks for a conflict between P and the current focus set. (Even if P just caused the removal of a focus word, P may still be in conflict with a different focus word.) If P conflicts with a focus word X, the priority of P is set to its rating reduced according to the inhibition strength of X, and the predict task goes to the rescheduling stage without making any predictions. If there is no conflict, the words contained in P are added to the focus set, and the normal predict task begins. These operations at the start of the predict task take care of adding words to focus, removing them, and adjusting prediction priorities.

The word task also has operations for focus by inhibition. A word set conflicts with a focus word X if any word from the set would be time-incompatible with X. For example, if X begins at position 40 and the word set left time is also 40, there is a conflict. In case of a conflict, the word set priority is lowered according to the strength of the focus word. At the start of the word task, the highest priority word set WS is selected. It may have become the highest priority in spite of a conflict with a focus word X. If so, WS is marked as 'immune' to inhibition by X. The word X is not removed from focus; however, if WS leads to the acquisition of a good word, a high priority prediction will remove X from focus later. Next, the word task checks for a conflict between WS and some focus word Y to which WS is not immune. If there is such a Y, the priority of WS is revised and the word-task is rescheduled. Otherwise, the normal word-task operations begin.

No other changes are involved in implementing focus by inhibition. The structure of the system makes it easy to try different methods for adjusting priorities, but, as mentioned earlier, the methods must be justified experimentally. This particular technique did not improve system performance, but perhaps a related approach will. The strength of the method is that it provides simple answers to how, when, and why to focus attention, while maintaining the completeness of the system control strategy. The weakness of this particular attempt is its inability to focus primarily on hits rather than on false alarms; perhaps greater selectivity in adding words to focus would produce a focus by inhibition with a positive effect on system performance (evidence suggesting that this may be so is given in Chapter IV, Section E.3.)

E. DISCUSSION

This section reviews the most significant features of our Executive System, compares our framework for speech understanding to some others, and sketches the evolution of our system over the last four years.

1. REVIEW

The primary functions of the Executive are system integration and control. The language definition is the principal mechanism for specifying knowledge source interactions, and phrases with their attributes and factors are the basic entities manipulated by the

Executive. Because of the central place given to the language definition, the Executive takes on the role of a parser in carrying out its integration and control functions. It builds a parse net data structure to hold intermediate results and hypotheses, and it uses the organization of the net to help eliminate wasteful duplication of effort. Two types of tasks interact to build the net: the predict task, which leads to predictions for words in the input, and the word task, which gets words and uses them to construct new phrases. The predict task operates in a top-down manner and ends by scheduling the word task; the word task operates in a bottom-up manner and ends by scheduling the predict task. Both tasks can operate bidirectionally through the input and are guided by a lookahead mechanism to avoid unnecessary operations.

The Executive controls the overall activity of the system by setting priorities. The fundamental data for setting priorities are factors based on both acoustic and linguistic information. Factors are combined to form scores and ratings. A phrase score reflects a quality judgment that is independent of the context of the phrase. For example, the score of a nonterminal phrase depends on the scores of its constituents and the factors that indicate how well the constituents go together, but it does not depend on higher level phrases that might include this one as a constituent. Because phrase scores are independent of context, they do not have to be recalculated for each possible use of the phrase. If a phrase gets a subthreshold score, it can be discarded without concern that in a different context it might do

better. Moreover, the language definition allows scores to be calculated for incomplete phrases, so the Executive has access to quality judgments from scores at each step as it adds constituents.

The score gives useful local information about a phrase, but in setting priorities we want to make use of global information concerning the sentential context; ideally, we do not want to waste time working on a phrase that is not part of the best current hypothesis about the entire utterance. In designing a method to make use of contextual constraints, we must consider the way the constraints are expressed. Our language definition is written in a style that represents a large part of the information procedurally rather than structurally. We use general structure declarations, with categories like noun phrase and verb phrase, and then use factor statements in the rule procedures to specify the detailed constraints. To get early and efficient access to the contextual information, we have developed a special technique for calculating phrase ratings. The rating of a phrase is intended to provide an estimate of the best score for an interpretation that can be constructed using the phrase. Phrase ratings are calculated by a heuristic search of the consumer tree for the phrase. The search is guided by previous ratings and by the results of executing consumer rules to gather factor information, and the phrase rating is determined by the score for the best complete consumer path constructed in the search. This technique provides the Executive with an effective way of estimating how well the phrase fits its possible

sentential contexts while avoiding an exhaustive search of the consumer tree. Experiments show that this context-checking method results in significant improvements in system performance.

Experiments such as the one just mentioned are important for evaluating a complex system design such as ours. It is not enough simply to demonstrate that a system with certain features can be implemented; a working system shows that the features are not disastrous, but it does not show what good effects, if any, the features have on performance. Experiments must be carried out in order to discover the actual effects and interactions of the design features. A useful method, which we have used in several experiments, is to evaluate system features by comparing the performance using a particular feature to the performance using a simpler alternative instead of that feature. The performance difference indicates the importance of the feature. Also, by testing different combinations of features, interactions between features are revealed. Using this method, we have carried out a large experiment concerning context checking, mapping all at once, island driving, and focus by inhibition. The results of the and other experiments regarding the system design are reported in Chapter IV.

The remainder of this section deals primarily with system integration and control in some other speech-understanding systems. The purpose is to clarify further the SRI system design rather than to give a complete review of the literature (for a good survey of recent speech recognition research, see Reddy, 1976). We consider systems developed

at Carnegie-Mellon University (CMU), at Bolt Beranek and Newman (BBN), and earlier at SRI.

2. CMU: HARPY AND HEARSAY-II

Two lines of speech research have been carried out at CMU in the last few years. One is based on the use of a simple dynamic programming model with all system knowledge represented in a state transition network. The other is based on a more complex design using multiple, cooperating knowledge sources. Both lines of research have been very productive, and there has been significant cross-fertilization between them. We discuss the most recent (and most successful) systems in each line: HARPY in the dynamic programming line and HEARSAY-II in the multiple knowledge source line.

The HARPY system has the best performance statistics of any existing system for understanding connected speech (Lowerre, 1976). The design for HARPY evolved from a comparative study of two previous CMU systems, Hearsay-I (see Erman, 1974a) and Dragon (see Baker, 1975). HARPY uses a state transition network to represent all possible pronunciations of all legal input language sentences. The network thus embodies the entire system knowledge of syntax, word pronunciations, and interword coarticulation effects. To process an utterance, HARPY looks for the path through the network that best matches the input. The search proceeds left-to-right, segment by segment across the utterance. At each segment, all paths are discarded that are more than a threshold

amount worse than the best path. This heuristic means that the system is not guaranteed to find the optimal path (the best one may be dropped if it starts out poorly), but the threshold for dropping paths is chosen so that in practice the optimal path is found in nearly all cases. Compared with an exhaustive search of all paths, there is a small sacrifice in accuracy for a large improvement in processing time. The performance achieved is impressive -- on a 1011 word vocabulary, HARPY got 92% correct of a test set of 100 sentences with an average of a little over 20 seconds of processing per second of speech on a 1.3 million instructions per second computer (DEC KL-10; these results were reported at a system demonstration at CMU on September 8, 1976). Because of its one-pass search strategy, HARPY has a low variance in its speed, which is also an important feature for a useful system. Finally, in contrast to most other speech-understanding systems, HARPY is conceptually simple and has been well-studied. For example, there is a good understanding of how its recognition time depends on parameters such as the number of samples in the input, the number of phonetic-classification templates, and the size of the recognition network.

HARPY achieves good performance with simple means, but it does so by sacrificing generality. With its reliance on a finite-state network for syntax, HARPY can only deal with very restricted input languages. The network representation also makes it difficult to deal with dynamic changes such as take place in natural connected discourse. For example, the use of anaphora and ellipsis depends on the preceding

sentences, and what is acceptable in one context may not be in another. The performance of HARPY appears to be largely dependent on its use of an extremely constrained input language. The average branching factor, the number of alternatives at each word, is about 9.5 for the grammar with which HARPY achieved the results mentioned above. Preliminary tests show a drop in accuracy to about 85% when the branching factor is increased to 25 (see Goodman et al., 1976). Based on measurements of our SRI language definition, we feel that a more natural input language with a 1000+ word vocabulary could easily have a branching factor well over 200 (see the discussion of our experiments in Chapter IV, Section D), so these results suggest that HARPY may be limited to very restricted languages unless there is a big improvement in acoustic accuracy. In addition, HARPY's strictly left-to-right search for a complete path through its network prevents it from dealing with incomplete sentences or inputs that in any way deviate from the predefined grammar. A more flexible control structure would allow it to build partial interpretations of such inputs from which an appropriate response could be inferred. In summary, on the limited tasks HARPY was designed for, it does very well. However, it does not represent or claim to be a general technique for speech understanding with a wide range of input languages. Undoubtedly, applications exist where a carefully constrained, artificial input language can be useful, and in these applications, HARPY offers a viable approach. For more general applications that may require a closer approximation to natural language input, the simple approach used by HARPY seems unlikely to succeed.

Another system developed at CMU, Hearsay-II (HS-II), has taken a more general approach (see Lesser et al., 1975). HS-II is based on many interesting design concepts. Perhaps the most distinctive is the representation of knowledge as self-activating, asynchronous, parallel processes that communicate with each other through a global data structure, called the 'blackboard'. There is an emphasis throughout HS-II on generality and uniformity in representation and control. The same approach is proposed for all levels of the system from signal processing to semantics. This search for generality is reminiscent of CMU efforts in other research areas such as problem solving. In fact, Lesser et al suggest viewing HS-II as a production system that is executed asynchronously.* They go on to state that in this uniform framework there are to be many small knowledge sources, each independent of the others. Knowledge source communication is to be through the generation and modification of globally accessible hypotheses on the blackboard. Changes on the blackboard are also to control the activation of knowledge sources. Each knowledge source is to have a precondition that is a description of some partial state of the blackboard. The knowledge source process can be run when the precondition is satisfied. The blackboard modifications made by one knowledge source trigger other knowledge sources by satisfying their preconditions. In addition to preconditions, each knowledge source has a specification of the kinds of changes it makes (information used in scheduling knowledge source activations), and a program which accomplishes those changes. There are

* See Newell (1973) for a discussion of production systems.

to be no separate data structures or state information for the individual knowledge sources; everything is to be done uniformly by means of the blackboard.

In the following discussion, we comment first on the HS-II goals and compare them to our system design. Later, we consider how close the actual HS-II system that was demonstrated in September 1976, approached the design goals given in the 1974 paper. HS-II as described above is an elegant and ambitious system. It is more general than HARPY, for instance, in that it does not force all knowledge into a state transition network representation. Also, it can hope to deal with sentence fragments that do not fit the predefined language. It is not limited to a left-to-right search and can build up phrases anywhere in the input in any order. HS-II and our system share the features just mentioned, but, in spite of that similarity, there are significant design contrasts between HS-II and our system with respect to system integration and control. First, unlike HS-II, we are not trying to use a uniform approach throughout the system for representing partial results and for controlling knowledge source operations. We feel it is very unlikely that a uniform method can produce satisfactory results for disparate operations such as semantic interpretation and acoustic labeling. Second, we emphasize explicit, direct knowledge source interactions through the procedural parts of rules rather than trying to keep all knowledge source interactions indirect through a global data structure. It is clear that some of the interactions must be indirect

to allow the Executive to determine dynamically the order in which tasks will be performed as it searches for an interpretation. This flexibility is necessary because of the inefficiencies associated with a fixed order of search (see Paxton, 1975, for further discussion of this point). We provide for indirect interactions by means of the phrases in the parse net and achieve modularity at the level of rules in the language definition. However, within a rule, the knowledge source interactions are explicitly stated in the procedure. There are several reasons for such a mix of direct and indirect interactions. One major consideration is to encourage interactions by providing a low-cost mode of communication. We feel that there is a large potential for mutual guidance that would not be realized if all knowledge source communication was indirect; the cost of modifying the global data structure and triggering the relevant preconditions would inhibit the interactions. In addition to being more efficient, direct interactions are often simpler to specify than the indirect ones. The simplicity further encourages the development of knowledge sources that cooperate closely. Another consideration is the inevitable overhead associated with scheduling and activating tasks. If every knowledge source interaction had this overhead, the system performance would suffer. Moreover, efficiency considerations would limit the algorithms for determining task priorities to simple operations such as merging local scores. By reducing the number of tasks, we are able to use a more complex scheduling algorithm and still keep the total scheduling cost small relative to the time spent actually performing the tasks. Tasks

in our system are thus substantial operations such as making entire sets of predictions or performing the acoustic processing needed to look for words at a particular location in the input. This relatively large task size is another difference between our system and HS-II as described in the 1974 paper. We also have a few large knowledge sources where they call for many small ones, and we have separate data structures for special use by the different knowledge sources rather than enforcing the uniform use of a single global data structure. Finally, we have developed techniques suitable for natural input languages, while HS-II has used word templates of a linguistically simple form (patterned after those developed for PARRY by Colby -- see Colby et al., 1974; see Hayes-Roth and Mostow, 1975, for a description of the HS-II template grammar).

In September 1976, a version of HS-II was demonstrated at CMU that was somewhat different in design from the description given in the 1974 paper. (The 1976 system is briefly described in Reddy et al., 1976). The 1976 HS-II has a few large knowledge sources rather than many small ones. One component does parameter extraction, acoustic analysis, segmentation, and labeling. Another does bottom-up word recognition based on syllables. A third is a word verification process using HARP techniques. This verifier checks words found by the bottom-up recognizer or words predicted by the syntactic component. A fourth knowledge source process is a word-pair adjacency tester. It looks at the speech data in a word-pair gap or overlap and decides if the pair is acceptable or not on the basis of the phonetic spellings and various

word juncture rules. A fifth knowledge source process is the word-sequence hypothesizer, which provides multiword seeds for an island driving control strategy. This process uses a bit matrix indicating allowable word-pairs in the language to ensure that the words in the multiword island are at least pairwise grammatically acceptable. It also calls the word-pair adjacency tester to make sure that the words in the island can be adjacent acoustically. The sixth and final knowledge source process is the parser. It has a template grammar for use in parsing word sequences, predicting words that can be syntactically adjacent to the ends of the sequence, and constructing larger sequences when predicted words are verified. The performance of HS-II on the same 1011 word vocabulary and input language as used in the tests of HARPY was 84% sentence accuracy (versus 92% for HARPY) and processing times of about 2 to 20 times longer than HARPY (results reported at a system demonstration at CMU on September 8, 1976). The poorer performance relative to HARPY is probably a result of the very restricted input language used for the tests; with such a language, HARPY's fast, simple techniques are adequate and give better accuracy by considering more alternatives before making a choice.

In many ways, the 1976 HS-II moved away from the description given in the 1974 paper and closer to our system design. As mentioned above, the demonstrated HS-II has a few large knowledge sources rather than many small ones. It makes use of direct, explicit knowledge source interactions: the word verifier is called directly from the bottom-up

word recognizer and the word junction tester is called from the word sequence hypothesizer. The knowledge sources maintain private data structures and state information rather than always using the blackboard, presumably because of the inefficiencies of doing everything in a uniform representation structure. As in our system, data-directed invocation is used for higher level processes but not for lower level ones. For example, the word verifier uses a HARPY control structure rather than the event-driven technique. Thus, the system uses varying control strategies rather than always using a precondition-activation scheme, again presumably because of the inefficiencies of the uniform method. Finally, along with giving the knowledge sources their own data structures and control structures, the size of the individual tasks performed by the knowledge sources were made relatively large to reduce scheduling costs.

All of the above changes bring HS-II closer in design to our system; however, differences still remain. For example, HS-II has adopted a multiword seed technique for island driving that apparently improves its ability to get started correctly. We have not tried that method, but it could be added to our system easily and might result in better island driving performance for us also. However, the technique may depend on having a restricted grammar to make the pairwise grammaticality tests sufficiently restrictive; with our more general language, such a test would be much less useful since so many word-pairs are possible. A more significant difference is the emphasis HS-II has

placed on parallel processing. The results of future experiments with HS-II on the special C.mmp multiprocessor system will be interesting as an indication of whether or not the potential parallelism can be effectively utilized (see Fennell and Lesser, 1975). On the other hand, we have emphasized natural input languages while HS-II has not. This emphasis has led us to develop special techniques such as context checking in setting phrase ratings. To the extent that the systems have converged, it indicates a growing consensus regarding system architecture for general speech understanding; the differences between the systems reflect the different research goals of the projects such as parallel processing in the case of HS-II and natural language processing in our case.

3. BBN: SPEECHLIS AND HWIM

SPEECHLIS and HWIM do not represent parallel speech projects in the way HARPY and HEARSAY-II do. Rather, the names refer to systems developed during two relatively distinct phases in the speech understanding research at BBN. In the first phase, from 1971 until 1975, BBN produced SPEECHLIS. In the second phase, 1975 to 1976, the BBN system was changed significantly in both its components and its overall organization. The changes merited the adoption of a new name, HWIM. The two systems are particularly different with respect to system integration and control, so it is appropriate for us to discuss both of them.

The SPEECHLIS system has an interesting design that evolved through the use of 'incremental simulation' combining computer programs and human simulators [see Woods and Makhoul, 1974, regarding incremental simulation; see the BBN papers in the 1974 IEEE Proceedings (Erman, 1974b) regarding SPEECHLIS]. In processing an utterance, SPEECHLIS starts with acoustic-phonetic analysis to produce a segment lattice representing all of the alternative segmentations of the utterance and the alternative phonetic identities of the segments. A lexical retrieval component then searches through the segment lattice for good matches for words of three or more phonemes. Such matches are added to a word lattice. A semantic component constructs sets of nonoverlapping words from the lattice by selecting semantically related words. These word sets, and information regarding their syntactic and semantic analysis, are called 'theories.' When semantic association can add no more words to a theory, the theory is passed to a syntactic component, called SFARSER -- "speech parser" (see Bates, 1975). SPARSER postulates grammatical structures for the words in the theory and proposes words to fill gaps between the words. The SPEECHLIS control component keeps track of different theories, proposals, and events (such as the retrieval of a word satisfying some proposal), and decides what to do next on the basis of a weighted sum of scores from lexical retrieval, syntax, and semantics.

SPARSER in particular is worth discussing in more detail. It uses an augmented transition network (ATN) formalism and a grammar that

was derived from earlier work at BBN (see Woods, Kaplan, and Nash-Webber, 1972). SPARSER is activated by the control component to process a set of nonoverlapping words, each contiguous word sequence being called an 'island.' The parser's job is to create parse paths through the islands and to predict syntactically acceptable words to fill the gaps between them. Each island is processed left-to-right. The first step is to find all the places in the grammar where the leftmost word of the island can occur. This and similar operations make use of precompiled grammatical indexes. The second step is to find all the transitions that lead to the arc for the first word but do not use the previous word of input (e.g., JUMP transitions). The grammar states that are reachable from the left of the first word by these lead-in transitions are used in making predictions for words to the left of the island. The third and last step in parsing an island is for paths to be extended through the island in preparation for making predictions at the right end. The parse paths are extended to the right in a best-first manner according to scores that reflect the likelihood of the arcs. If no paths can be found through the island, the theory is rejected. Otherwise, the ends of the paths determine states for predictions to the right of the island. (Notice that predictions on the right correspond to the end of a path leading completely through the island, but predictions on the left are constrained only by the leftmost word.) After all the islands in the theory have been processed, SPARSER looks for predictions to fill the gaps. In particular, it looks for small gaps that could be filled by a single word. If a prediction is made

from both sides of such a gap, a proposal is made for the lexical retrieval component to search for the predicted words. While SPARSER is making proposals, it can return to an island and try to extend more paths through it to get more predictions on the right. This is done in hopes of producing a common prediction to fill a gap.

The ability to use pairs of islands to make predictions to fill gaps is one of the most interesting features of SPARSER. Miller's LPARS (see Miller, 1973) also made predictions to fill gaps between islands, but it used an exhaustive search strategy and so was probably impractical for large vocabularies and grammars. SPARSER appears to have been the first to provide a combinatorially feasible control strategy for multi-island processing. Another noteworthy feature of SPARSER is the large amount of merging of alternatives resulting in the sharing of information among different theories. For instance, only one instance of a particular state and input-location configuration is ever created and it is shared by all theories. However, there are problems in the SPARSER design. First, there is not enough communication with other components, especially semantics. For example, SPARSER's proposals are not constrained by semantic information and so they may lead to wasted effort looking for words that are acceptable syntactically but bad semantically. Second, the use of multi-island theories probably costs more than it is worth. There are no inter-island consistency checks; the only interaction among islands is in making predictions at gaps that are small enough to be filled by a

single word. Even when there is such an interaction, it is simply to look for common predictions, and failure to find one does not disqualify the theory since the gap might be filled by more than one word. The overall result is that there is relatively little gain from having multi-island theories.

In contrast to the small gain from having multi-island theories, the costs are significant. The same island may occur in many different theories, and, although SPARSER does share information among theories, there is a nontrivial overhead for reprocessing an island in each theory.* A final and especially serious problem with SPARSER is its failure to make good use of the available grammatical constraints to limit its operation. We mentioned above that predictions on the left of islands depend only on the leftmost word and are therefore not taking advantage of possible constraints provided by the other words in the island. Another failure to use the available information relates to the restrictions on arcs. The grammar operations on arcs are explicitly divided into local ones, which depend only on the word or constituent for the particular arc, and context sensitive ones, which depend on information from some other arc. SPARSER performs local operations when it creates arc transitions, but it does not do any context sensitive operations until it has a complete path through a network allowing a constituent to be formed. When such a complete path is acquired,

* The time for reprocessing is about one-third the time for initial processing; Bates (1975, p.111) reports 16.5 CPU seconds for reprocessing a theory that took 47.5 seconds to process originally.

SPARSER goes through the path left-to-right executing the context sensitive operations, thus making sure that the required information is available when it is needed. This is a simple solution to the problem of context sensitive operations, but it fails to prevent SPARSER from working on partial paths that are bad syntactically. Bates acknowledges this problem and comments to the effect that the parser could take context into account more easily if the grammar had less of a left-to-right orientation (Bates, 1975, p.190). This comment supports our decision to use a language definition representation that is more neutral than ATNs with respect to control strategies.

SPEECHLIS has other problems in addition to those mentioned above regarding SPARSER. First, the use of semantic associations does not seem to be sufficiently selective to help in producing a few good starting theories for the system. In a limited task domain, most words will be semantically related, so almost any set of words found by lexical retrieval will be accepted by semantics.* If lexical retrieval is very accurate and only finds a few outstanding matches, the poor selectivity of semantics will not hurt, but semantic association will be an unnecessary step. However, if lexical retrieval produces many words that match equally well, semantic association seems likely to swamp the system with theories. Given the problems with acoustic recognition, the

* However, Nash-Webber and Bruce suggest that in the lunar rocks task domain used in SPEECHLIS, the possible semantic relationships among the entities were so limited that this was not a problem (Woods et al., 1976b, p.42). Apparently it became a problem when the task domain was changed to travel budgets (see comments in Woods et al., 1975b, p.44).

latter alternative seems more probable. A second problem with SPEECHLIS is the generally poor communication among knowledge sources. We mentioned that syntax makes proposals that may be bad semantically. The converse is also true; semantics makes proposals that may be bad syntactically. In general, any component in SPEECHLIS is free to make proposals, but there is inadequate communication to ensure that such proposals are mutually satisfactory. There is a need for closer cooperation among the components.

The performance of SPEECHLIS seems to support these criticisms. There has been no report of systematic tests of its performance, but comments appear in various publications (especially Nash-Webber and Bruce in Woods et al., 1976b; also in Woods et al., 1975b). The processing time required by SPEECHLIS may have prevented extensive testing. For example, SPARSER reportedly can take over 40 seconds of processing on a single theory if much bottom up processing is necessary (Bates, 1975, p.111). It is difficult to do much testing at that rate. Also, storage demands apparently made it impossible to run tests to completion in many cases. There is a comment (in Woods et al., 1975b, p.44) that, as a result of space problems during the semantic association phase, the system was unable to complete processing any utterances. (The comment referred to tests with the travel budget task, which may have had worse space problems for semantic association than the previous lunar rocks task.) By 1976, SPEECHLIS was replaced at BBN by a new system called HWIM ("Hear What I Mean"). In the change, the

control strategy was replaced, the semantic association component was dropped, and SPARSER was dropped.

The HWIM system is in several respects a reaction to the problems of SPEECHLIS. For instance, it attacks the problems related to coordinating knowledge sources by embedding much of its syntactic, semantic, and pragmatic knowledge in a transition network representation. The result is called a 'pragmatic grammar' and has much in common with the 'task oriented grammars' of HARPY and Hearsay-II. General categories like noun phrase and verb phrase are replaced by task specific ones like 'meetings,' 'trips,' and 'expenses' (Woods et al., 1976b, p.54). The grammar has more states and arcs than the more general SPEECHLIS grammar because general categories have been split into special ones and also because many word arcs have been added. Like the CMU template grammars, large portions of the BBN pragmatic grammar would have to be rewritten for a different task domain (as they admit; Woods, et al., 1975b, p.23).

Another problem in SPEECHLIS was SPARSER's failure to use context sensitive restrictions until it had a complete constituent. HWIM has a partial solution to this problem based on the specification with each arc operation of the scope of its context dependency. For example, if arc A has a test that depends on a feature set by an action on arc B, the test on A is marked to show that its scope includes B. The HWIM parser executes context sensitive actions as soon as the parse path covers the necessary scope. This method is an improvement over

SPARSER, but it still is not as thorough in its use of contextual information as our technique of exploring the sentential context to set ratings. HWIM is better than SPARSER because it does not wait to complete a constituent before testing the relations among its subphrases. However, HWIM does wait until it has acquired complete phrases for the arcs in the scope of the operation, and often it is not necessary to wait that long. The tests often depend on phrase attributes that are determined long before the phrase is complete. To use an example from our system, the number attribute of a noun phrase can frequently be inferred from its determiner without waiting for the entire phrase to be built. If the sentential context calls for a singular noun phrase but the determiner indicates a plural one, our method of context checking recognizes the inconsistency when it sets the rating for the incomplete noun phrase, whereas the HWIM method would not notice the conflict until it had a complete noun phrase and tried to use it in the sentence phrase.

A third problem in SPEECHLIS was inefficiency caused by theories with multiple islands. The same island could occur in many different theories resulting in substantial duplication of effort. HWIM resolves this problem by limiting theories to a single island. Duplication of effort is reduced by this method, but not eliminated. There is still a problem of forming the same island in many different ways depending on the order in which words are added. HWIM reduces this problem somewhat by a technique called 'island collisions'. Before

explaining this technique, we sketch the HWIM lexical retrieval component that plays a vital role in the system.

HWIM depends on the existence of an efficient and effective lexical retrieval component. In providing one, Klovstad has produced a particularly interesting part of the system (see Klovstad in Woods et al., 1976b). Significant features of the lexical retrieval program include the following:

- * It finds the n best matching words without requiring exhaustive testing of all the words in the vocabulary. (The processing time varies with the log of the vocabulary size rather than linearly.)
- * It can take advantage of syntactic predictions to constrain its search for matches.
- * It makes effective use of phonological word boundary rules by precomputing their possible effects. This technique eliminates the need to make very loose judgments at word boundaries to compensate for possible coarticulation effects.

The internal dictionary for lexical retrieval is stored as a tree structure merging common phonetic sequences. A tree with initial sequences merged is used for left-to-right searches; another tree merging final sequences is used for right-to-left searches. Word boundary rules are applied to the trees to reflect the possible coarticulation effects. To allow selective retrieval according to syntactic category, each node in the dictionary tree is tagged with a bit mask showing the categories of words in that branch. The lookup procedure matches paths through the dictionary tree against phonemes in

the segment lattice. When a path reaches the end of a word spelling, a match has been found. The matching operation allows for 'merges' and 'splits' to take care of possible segmentation errors. In a merge, a single phoneme accounts for two adjacent acoustic segments. In a split, two adjacent phonemes map onto a single acoustic segment. Path scoring penalizes matches that require splits or merges. Paths are eliminated from consideration by three operations: (1) syntactic selection -- if no words in the selected syntactic categories are reachable by the path, eliminate it; (2) forward pruning -- if the path is judged unlikely to produce words with scores better than some threshold, eliminate it; and (3) finite memory -- if the path score falls relative to the others so that the path is no longer among the k best paths (where k is a parameter), eliminate it.

The lexical retrieval component was tested on 99 sentences with a dictionary of 702 words. Directed to return up to 15 matches per sentence, it selected an average of 9.48 distinct matches per sentence. Overall, 23.7% of the words returned were correct, and in about 70% of the sentences the best match was correct. In only about 4% of the sentences did the lexical retrieval process fail to find at least one correct word.* The processing times for these unanchored scans is

* Klovstad suggests (in Woods et al., 1976b, p.106) using the average ratio of correct to incorrect words as a measure of lexical retrieval performance. He reports a ratio of 0.3112; however, this is actually the ratio of the average number of correct words per scan (2.25) to the average number of incorrect words per scan (7.23). The average of ratios is not equivalent to the ratio of averages, so Klovstad is apparently suggesting one measure and reporting another.

reported to be about one CPU minute for a 2.5 second utterance (Nash-Webber and Bruce, in Woods et al., 1976b, p.46). They also report that to do a search at a given position in the segment lattice with a 448-word vocabulary takes an average of 6.83 CPU seconds (Woods et al., 1976b, p.49). We are especially interested in the lexical retrieval component because it presents an efficient alternative to our map-all-words-at-once control strategy.

The HWIM island driving control strategy makes heavy use of the lexical retrieval component. HWIM starts processing an utterance by creating a segment lattice. Lexical retrieval then does an unanchored scan to find up to 15 of the best matching words. A one word theory is created from the best match, and the other matches are reserved for later use if the first one runs into trouble. There is no attempt to use semantic associations to form multiword theories as was done in SPEECHLIS. Instead, syntax is immediately called with the theory to make predictions to extend it. New theories are formed by adding words to old ones or by starting another one-word theory using one of the original words from lexical retrieval. There is little sharing of information among islands. One exception occurs if two islands 'collide' by growing together. In this case, the words from the two islands are joined together in a single operation. Other than this, the different islands do not interact, and there is none of the merging of work on common subparts that we have in our parse net or that SPARSER had in its shared configurations and transitions. By giving up sharing,

the parsing becomes simpler, but there are combinatorial problems if a large number of islands must be considered.

In the areas of scoring and priority setting, HWIM is more sophisticated than SPEECHLIS. HWIM uses experimentally determined statistical scoring. The use of probabilities provides a uniform scoring technique so that scores from different sources can be compared and combined in a theoretically sound manner.* On the basis of these scores, HWIM uses a 'shortfall density' technique for setting priorities. This technique depends on having a lexical retrieval component that can find the best matching words spanning the input. The shortfall method begins by finding the best spanning words and uses them to set an upper bound score for each segment of speech. The 'shortfall score' for an island is the difference between the sum of the upper bounds for segments in the island and the actual scores for words in the island. (Scores are additive in HWIM.) The 'shortfall density' for an island is its shortfall score divided by its length. The priority for working on an island is determined by the island's shortfall density: the smaller the shortfall density, the higher the priority. The control strategy using shortfall density priorities is guaranteed to find the optimal interpretation as its first spanning island. In the terminology of heuristic search, this is an 'admissible' strategy (see Hart, Nilsson, and Raphael, 1968). However, the price of admissibility for this method appears to be a relatively breadth-first search. BBN has

* The change to probabilistic scoring began during the end of the SPEECHLIS period at BBN (see Klovstad in Woods et al., 1975a, pp.33-39).

explored a number of variations to try to get better performance, and, as of September 1976, their favorite variant sacrifices the guarantee of finding the optimal island first. It restricts the island seeds to be near the left end of the utterance. After a seed word is selected, the island is first extended to the left boundary of the utterance and then to the right. This method results in primarily left-to-right processing, so there is less chance of duplicating effort by creating the same island in different ways. It avoids the potential problems of a strictly left-to-right method because it can jump over the start of the sentence to pick a seed word. This compromise may offer a way around some of the problems with island driving that we report in the next chapter.

To summarize, HWIM has introduced a variety of interesting design concepts. In particular, we refer to the work on lexical retrieval, island parsing, probabilistic scoring, and shortfall density priorities. However, most of these concepts still need to be tested to determine their effect on the performance of the system. In our own work, we have discovered that intuitively appealing design features can sometimes have distressing effects on actual performance.

Some final comments are required regarding the use of a 'pragmatic' grammar in HWIM. Nash-Webber and Bruce state that the overall performance of HWIM is improved by fusing syntactic, semantic, and pragmatic knowledge sources, and that the improvement "comes from being able to constrain as soon and as tightly as possible the

acceptable ways in which a given theory can be extended" (Woods et al., 1976b, p.53). They go on to say that "a naive conception of KS [knowledge source] interaction, which assumes that if communication channels exist, they will be used effectively, is wrong, at least in terms of currently realizable systems of HWIM's size and complexity" (Woods et al., 1976b, p.55). On this basis, they propose a principle that they dub "WORK TOGETHER":

If it is found that one must frequently consider simultaneously information from several KSs, then the activity of those KSs should be tightly coupled. (p.55)

We agree that tight coupling of knowledge sources is important; we have emphasized that fact in the design of our systems since we began working on speech understanding (see, for example, comments in Walker, 1973). We are willing to believe that pragmatic grammar improves HWIM's performance (although Nash-Webber and Bruce offer no evidence supporting the claim). It may also be true that the improvement comes from the source they suggest -- namely, from being able to apply constraints as soon and as tightly as possible (again, no evidence is given). An alternative explanation is that the pragmatic grammar improves performance by greatly reducing the generality of the input language: fewer choices, so better results. However, if Nash-Webber and Bruce are correct in their claim that the improvement is caused by better use of constraints, then the improved performance can be attained without resorting to a pragmatic grammar. In our system,

explicit checking of the sentential context as part of setting ratings leads to early and effective use of constraints from all relevant sources of knowledge. Constraints are actually applied earlier by our technique than in HWIM since we do not require completing constituents before considering their interrelations.

The use of a pragmatic grammar may improve efficiency, but at a cost of increased size and complexity and decreased generality. Using our techniques, knowledge sources can be used effectively without being fused in the BBN manner. We get tight coupling of knowledge sources and early, thorough application of constraints without giving up the integrity or generality of the different knowledge sources.

4. EARLIER SRI SYSTEMS

It may be useful to give a brief sketch of the evolution of our system design and mention some of the techniques that were tried and then modified or discarded. The emphasis of the sketch is on unpublished material, but we also review work discussed more fully in earlier publications.

Our 1972 system (see Paxton and Robinson, 1973) tried to minimize the demands on acoustics by restricting the acoustic processing to testing words that had been hypothesized on the basis of other knowledge. The hope was that by hypothesizing words in roughly the order of their likelihood in a particular context, we could reduce the

average number of errors in acoustic recognition. To achieve flexibility in ordering hypotheses, the system used a best-first strategy rather than depth-first with backtracking. The language definition was written in a procedural style following Winograd (1971; in fact, an earlier version of our system had been constructed by making modifications to Winograd's SHRDLU; see Walker 1973). There was some effort to share information among different attempts to parse an input, but sharing was limited to successes rather than failures or partial results. Rating of alternatives was done by associating priority functions with alternatives at choice points. The original intention was to implement the system using 'spaghetti stacks' (see Bobrow and Wegbreit, 1973), but since that facility was not available in time, an interpreter was written instead.

During 1973 we increased the amount of interparse cooperation in the system by introducing a limited version of Kaplan's producer-consumer scheme (Paxton, 1975; also see Kaplan, 1973b). It was restricted to a single level of producers; in other words, the producers could not be consumers too. Also, the context dependency within producers was restricted to the lexical level. Global control was still embedded in the procedural grammar. Performance results for this system are reported in Walker (1974, 1975). With a 54 word vocabulary, the system got 44 of 71 sentences correct (62%).

In 1974 we began working with SDC on a joint project. This change led to a major redesign influenced by SDC's previous work (see

Barnett, 1973; Ritea, 1974) and the new Hearsay-II design (Lesser et al., 1975). The SDC parser was able to work top-down, bottom-up, and bidirectionally, and we decided to provide that much flexibility in our new system also. This decision required a change in the language definition methodology and resulted in the adoption of augmented phrase structure rules. From HS-II, we got the idea of distinguishing between ratings (reflecting 'goodness') and priorities (reflecting 'importance'). However, we still thought of focus of attention in terms of suspending and reawakening processes and had trouble specifying why, when, and how, such operations should take place. By the end of the year, we replaced the ideas of suspending and reawakening by the idea of changing priorities and designed focus by inhibition in essentially its current form. The original version used entire phrases for focus, but when that resulted in a great number of incorrect phrases in focus, we shifted to focusing on words selected from phrases. Also during this period, discussions with Barnett at SDC led to the ideas of phrase mapping and lexical subsetting, the former to deal with coarticulation effects and the latter to give the system more guidance from acoustics.

By late 1974, a system was implemented based on augmented phrase structure rules and a general producer-consumer structure for parsing (see sections II and III in Walker et al., 1975). However, the rules did not allow options or alternatives, and the attributes and factors were defined by simple lists of assignment statements. The producer-consumer structure for the parse net took care of cooperation

among different attempts to parse an utterance, but it increased the problem of using contextual restrictions in setting priorities. From the beginning, we felt we had to take the context into account, but our first effort was a failure. Rather than searching up the consumer tree as we do now, we tried passing restrictions down the tree from consumers to producers. To illustrate, let C be a consumer phrase with an acquired constituent A and a missing constituent B. A factor expression for C is in general a function F of attributes of A and B. A new function F' can be formed by fixing the arguments of F that depend on A to the actual values from A; thus, F' depends on attributes of B only and can be passed down to B-producers as a restriction from C. Similarly, B-producers can further modify the arguments of F' to yield restrictions to pass down to their producers. This method was implemented and debugged, but as soon as it was tried on tests simulating speech input, its extravagant use of storage for propagating restrictions showed that it would have to be replaced.

The system design steadily evolved during 1975. The tree search context checking technique was developed for setting ratings. The first version searched the entire consumer tree to find the best path; later, the current technique was developed to do a heuristic search for a near optimal path. The system had a relatively small task size; for example, the prediction task added only the immediate producers for the highest priority phrase rather than adding the entire subnet of producers. The small task size was intended to give a great

deal of control over the operation of the system; it did that, but it also caused the system to spend over half of its time calculating and recalculating priorities. By adopting a larger task size, the scheduling overhead has now been reduced to about 12% of the Executive processing time. Another problem was the lack of alternatives and options in rules. Without them, it was impossible to merge related rules. For instance, there were 10 S rules, 5 of which had initial NPs. There were five empty S phrase consumers to consider when setting ratings for initial NPs, and whenever an initial NP was found, five incomplete S phrases had to be constructed. After we redesigned the language definition using rules with alternatives and options, there was only one S rule with an initial NP. The use of alternatives and options in rules complicates the Executive, but it provides an important reduction in the number of rules and hence in the processing and storage requirements of the system.

Also during 1975, we added multiword lexical entries, lookahead, word task scheduling by lexical subcategories rather than by individual words, and gap/overlap testing by syllables rather than by fixed duration. To improve efficiency, we began to use a preconstructed initial parse net and preconstructed empty phrases. The monitor subnet for island driving was added as more than just an efficiency measure; it provides the necessary consumer context for island driving so that all consumer paths lead to a root-category phrase.

In late 1975 and early 1976, we put together a version of our system on the SDC IBM 370/145.* Before thorough testing could be done, SDC's computer was removed for administrative reasons, and our joint effort with SDC came to an end. However, enough information about their acoustic processing routines was collected so that we could test our parts of the system on our own computer by using a simulation. The results of those simulation experiments are reported in Chapter IV.

* Ann Robinson was primarily responsible for transferring the system from our PDP-10 to the SDC IBM/370.

IV EXPERIMENTAL STUDIES

Prepared by William . Paxton

CONTENTS:

- A. Introduction
- B. Experiment 1 -- Mapper Performance
- C. Mapper Simulation
- D. Experiment 2 -- Fanout
- E. Experiment 3 -- Control Strategy Design Choices
 - 1. Accuracy
 - 2. Runtime
 - 3. Review of the Effects
- F. Experiment 4 -- Gaps and Overlaps
- G. Experiment 5 -- Increased Vocabulary and Improved Acoustics
- H. Detailed Measurements of Executive Operation
- I. Conclusion
- J. Test Sentences

A. INTRODUCTION

This chapter discusses a series of experiments concerning our speech-understanding system. Information regarding the acoustic processing is reported in the first experiment. As well as being of interest in its own right, this information was used in simulating the acoustic processing for the other experiments. The second experiment deals with the 'fanout,' the number of alternatives at each word, both for the language alone and in combination with the acoustics. Fanout provides a quantitative measure of the difficulty of the speech-understanding task. In the third experiment, the main experiment of the

series, the standard speech system is tested on a set of 60 sentences for all combinations of four control strategy design choices. The best configuration from Experiment 3 is tested again in the fourth experiment, allowing different sizes of gaps and overlaps between words in the simulated acoustic processing. The fifth experiment studies the performance of the two most promising system configurations from Experiment 3, while varying vocabulary size and acoustic processing accuracy. The final study deals with detailed measurements of the Executive's performance for the best version of the system on Experiment 3. The following sections assume familiarity with the Executive System, at least to the level of detail given in the overview (Section C) in Chapter III.

B. EXPERIMENT 1 -- MAPPER PERFORMANCE

The first experiment deals with the performance of the system component called the 'mapper' (described in Chapter I, Section B, and in Bernstein, 1975). To review, the mapper carries out acoustic tests: given a predicted word and location in the input, the mapper either rejects the word or accepts it and reports its beginning and ending boundaries rounded to the nearest 0.05 second. If the word is accepted, the mapper also gives it a score between 0 and 100, indicating how well it matches the input (100 indicates a perfect match). Words accepted by the mapper are either 'hits', words really in the input sentence, or false alarms, words accepted although not in the input.

The mapper was tested by calling it for all of the words in the vocabulary at the start of an utterance and then at each position where a previously accepted word ended (independent of whether the previous word was a hit or a false alarm). This procedure resulted in testing the entire vocabulary at an average of about 16 out of the 20 possible positions per second of speech (recall that word boundaries are rounded to multiples of 0.05 second, so there are 20 possible ending positions per second).^{*} Overall, tests were made at 160 positions in 11 test utterances. For the 305-word vocabulary used in the following experiments, the mapper had 48 hits and 1564 false alarms. The false alarms were distributed throughout the vocabulary [229 of the 305 words (75%) were falsely accepted at least once], with small words like "a" and "the" each accounting for more than 30 false alarms.

The false-alarm rate for the mapper was determined by counting the number of false alarms that fell entirely within a section of the input. For the 305-word vocabulary, the average rate was 114 false alarms per second of speech. Since there were about 3 hits per second of speech, this rate indicates that the mapper produced an average of almost 40 false alarms for each hit. Figure IV-1 summarizes the results for three vocabulary sizes.

^{*} This experiment was originally designed to record the results of all mapper calls that might be made in a left-to-right parse. The intention was to use this information in place of the mapper in tests of the entire system. However, technical and administrative difficulties made it impossible to gather enough information to satisfy the original goal. If the original goal had been to provide data for a simulation of the mapper, the mapper would have simply been tested on the entire vocabulary across each utterance at 0.05-second intervals. The change in goals may have resulted in missing some potential false alarms because of the untested positions where no word ended.

	VOCABULARY SIZE (words)		
	305	451	823
False Alarm (FA) Rate	114	142	360
Mean FA Score	59	58	58
Total FAs in sample	1564	2026	4989
Words with FAs	229	326	654
Words with no FAs	76	125	169

Figure IV-1. MAPPER PERFORMANCE

As partial compensation for the high false-alarm rate, there were no 'misses' (cases in which the mapper failed to accept a correct word), and the mean hit score was higher than the mean false alarm score (73.5 versus 59.4), although both score distributions spread over the entire range, from near 100 down to the threshold of 45. The score distributions are shown in Figure IV-2.

The cumulative percentage distributions for the scores are shown in Figure IV-3. Note that a threshold of 55 instead of 45 would eliminate 45% of the false alarms but only 6% of the hits. We are not suggesting such an increase in the threshold, but it illustrates the extent to which the false alarms are found at low scores.

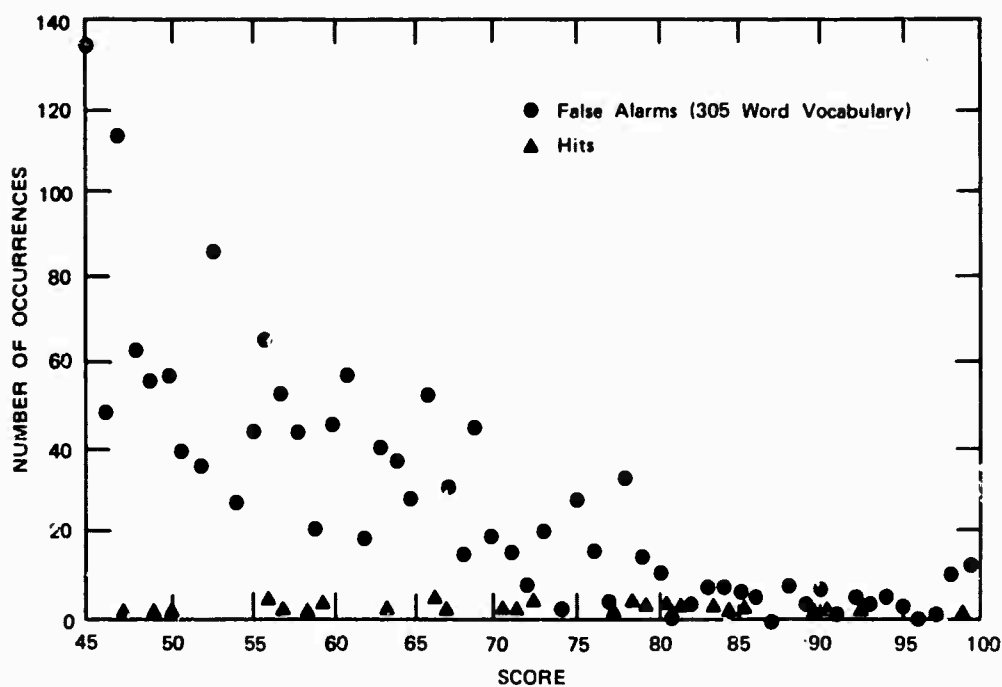


FIGURE IV-2 SCORE DISTRIBUTIONS FOR FALSE ALARMS AND HITS

C. MAPPER SIMULATION

The following experiments use a simulation of the mapper based on the data gathered in the first experiment. To simulate the performance of the mapper on a particular sentence, the words of the sentence were first assigned lengths in seconds of speech. Each word was then assigned a score picked at random from the total collection of hit

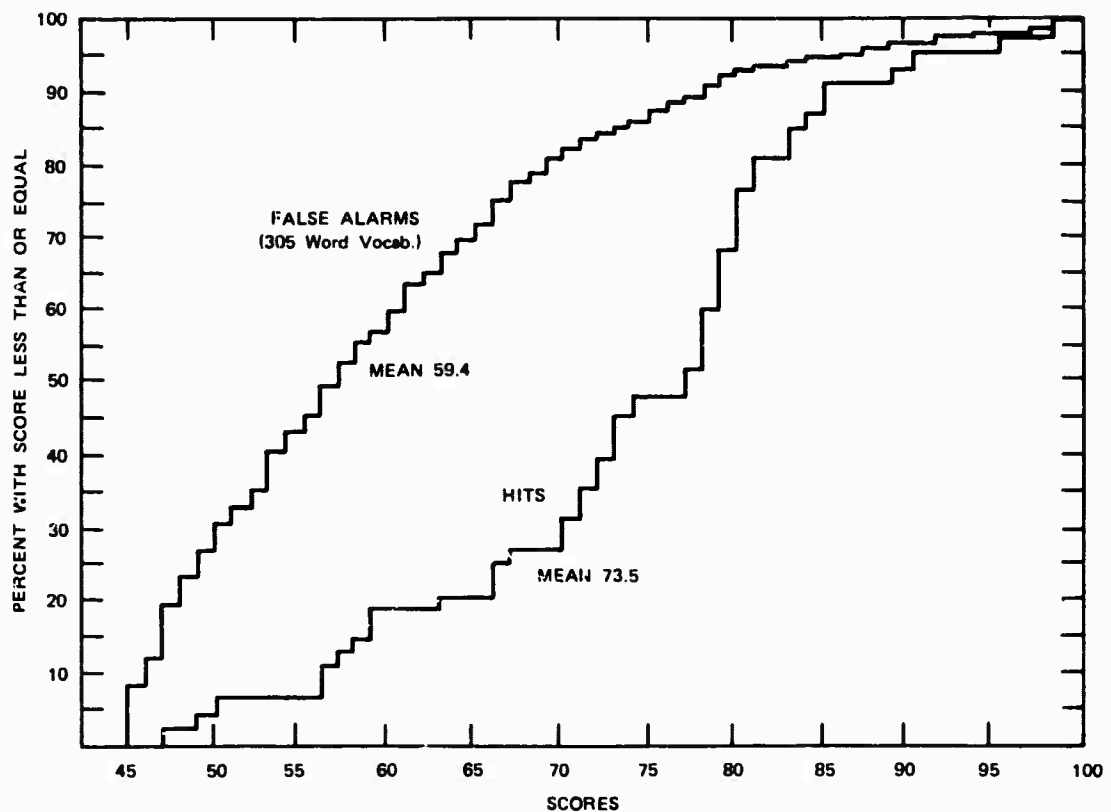


FIGURE IV-3 CUMULATIVE DISTRIBUTIONS OF HIT AND FALSE ALARM SCORES

scores actually produced by the mapper. The words were concatenated to determine the length of the utterance, and the length was multiplied by the false alarm rate (114 false alarms per second of speech) to give the total number of false alarms to be simulated. The false alarms were selected randomly from the 1564 false alarms produced by the mapper, and then positioned randomly in the sentence.

In computing the simulated processing time for the mapper in later experiments, we used figures of 0.30-second processing per word tested, 1.0 second per position for lexical subsetting,* and 10.0 seconds per second of speech in the sentence if 'island driving' was being simulated (see discussion of Experiment 3). These timing figures are based on rough measurements of the mapper running on an IBM System/370 Model 145.

This simulation reproduces the observed mapper performance statistics for the false-alarm rate, the hit scores, and the particular false-alarm words and scores. Because of insufficient data, we cannot use a simulation that reproduces more complex statistics such as the co-occurrence of various hits and false alarms or possible dependence of scores on the position within the utterance. We cannot say with certainty that these more complex statistics are unimportant, and, consequently, we do not claim that the particular performance levels in the following simulation tests will be precise estimates of the system's actual performance with a real mapper. In view of this limitation, the following experiments that use the simulation are designed to emphasize comparisons between performance levels rather than basing judgments on absolute performance levels. For example, to judge some design feature F, we look at the difference in performance with F versus without it,

* Recall that the lexical subsetting component uses local, robust acoustic cues to select a subset of the lexicon for further testing at a particular input location. To simulate this component, the system creates a subset containing the simulated hits and false alarms at or near the specified location and then adds randomly selected words to increase the size of the subset to a specified value. For the 351-word vocabulary, the subset size was set at 50 reflecting the expected performance of a well-tuned version of the lexical subsetting component.

rather than simply reporting the absolute performance observed with F. Such a comparative experiment is always appropriate as a way to judge the effects of a design feature, but it is particularly important when simulating a major component of the system. For instance, if some property P of the actual mapper is not reflected in the simulation, that lack may affect the performance levels of the versions of the system with and without some design feature F. However, as long as both versions are affected in roughly the same way, conclusions drawn from significant differences in performance in the simulation tests will probably be valid regarding performance with the actual mapper. In short, the simulation experiments should be good for making design decisions based on comparative judgments, but the absolute performance levels in the experiments must be taken with a grain of salt.

In spite of this limitation, there were compelling reasons for doing simulation experiments rather than testing the system with the actual mapper. First, it would have been impossible to do extensive testing with the actual mapper -- the time required would have been too great, both because of increased processing time and because of increased memory demands (leading to large delays for 'page swapping' by the time-sharing system). By using a simulation, we were able to do a large number of tests because the processing and memory demands of the mapper were eliminated. Also, we were able to study interesting control strategies, such as mapping all at once, which would be too slow for use with the actual mapper, but which otherwise have good effects on system performance and suggest new system designs.

Another reason for doing simulation experiments was that our access to the actual mapper was cut off when the SDC speech project computer was removed in March 1976. (This reason is certainly not a scientific one, but it illustrates the fact that not all decisions in research are determined by scientific considerations.) Luckily, there was a week between the time that the mapper started working well enough to be tested and the time that the SDC computer was removed. During that week the first experiment was performed. The other experiments were carried out on the SRI computer using a simulated mapper.

D. EXPERIMENT 2 -- FANOUT

The second experiment deals with the fanout in the language, with and without acoustic constraints. 'Fanout' is defined as the number of words that can be successfully appended to an initial substring of some sentence, to produce either a complete sentence or a string that can potentially be completed to form a sentence. The average fanout over a large number of initial substrings provides a measure of the uncertainty of each word, as indicated by the number of alternatives open to the system.*

The fanout was measured for 11 sentences, together containing a total of 67 words. The fanout was measured only for initial substrings

* Goodraan (1976) considers a variety of measures of language complexity. Our fanout measure roughly corresponds to his 'dynamic branching factor'. However, his methods deal only with finite-state grammars, so the correspondence between the measures is not exact.

of the actual sentences; it was not measured along false paths. The distribution of the size of the fanout was roughly bimodal (see Figure IV-4). Using the 305-word vocabulary and ignoring acoustic constraints, 24 positions (36%) had a fanout of less than 30 words, while 33 positions (49%) had a fanout of more than 173 words. The small fanout positions were places allowing only vocabulary classes with a small number of members, classes such as preposition or verb. The large fanout positions corresponded to places where a noun could be expected. The mean fanout was 117, with a standard deviation of 90 and a maximum of 219. The fanout at the beginning of sentences was 206; the average fanout at noninitial positions was 100.

The fanout with acoustic constraints is based on the simulated mapper data. It is calculated by counting the number of words that are accepted by the simulated mapper at a position starting plus or minus 0.05 second from the end of an initial substring of hits, and that are also in the fanout set without acoustic constraints for that substring. In addition to recording the size of the fanout, we ordered the set of words by increasing mapper scores and computed the rank of the hit. For example, if two false alarms had scores higher than the hit, the rank of the hit would be three. For the 305-word vocabulary, the mean fanout with speech was 18, the hit had the best score in 28% of the cases, and the average hit rank was 3.7. The fact that the hit rank is much smaller than half of the fanout reflects the previously mentioned difference between the score distributions for hits and false alarms.

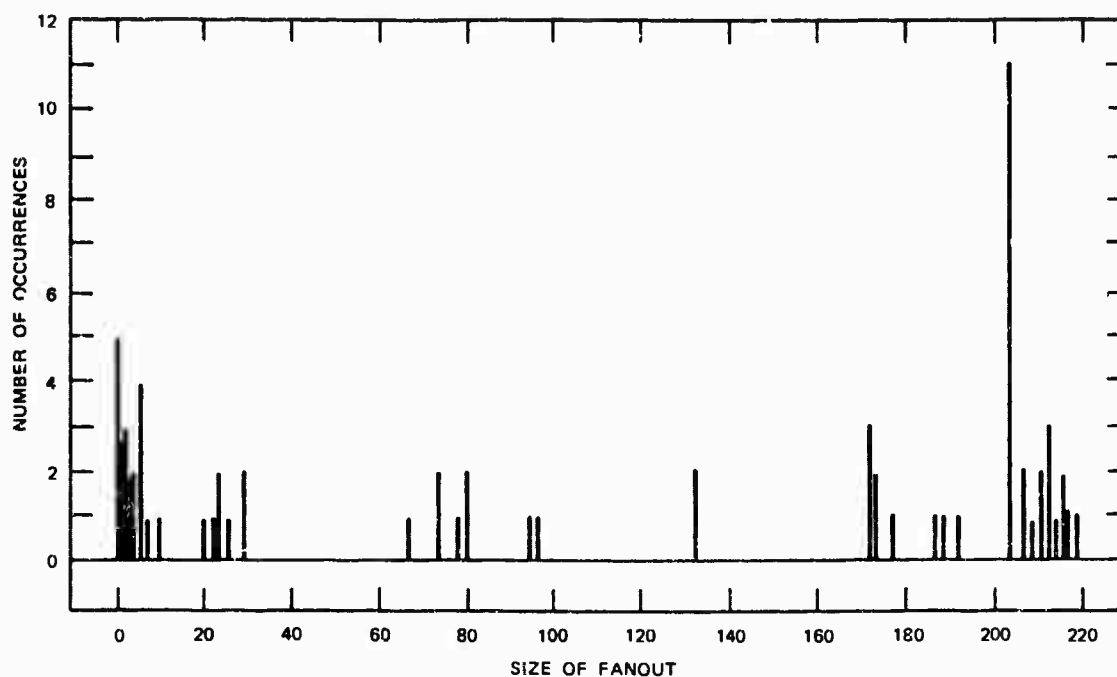


FIGURE IV-4 FANOUT HISTOGRAM

The results of this experiment help to show why the control strategy problem for speech understanding is so difficult. The results suggest that, on the average, there will be between two and three false alarms with higher scores than the actual hit to tempt the system down false paths. Also, the hit had the best score in only 28% of the cases, which appears to imply that the probability of correctly answering a sentence n words long is 0.28 raised to the n th power. However, the

accuracy of the system in the following experiments is actually much higher than that, so there must be compensating factors tending to bring the system back to the correct path. For example, the fanout following a false alarm is probably smaller than the fanout following a hit, causing false paths to be dead ends. The decrease in fanout should be most pronounced near the boundaries of an utterance, where many words are eliminated because their minimum duration is greater than the available time. Similarly, false paths may be impossible to complete because too much speech remains. For instance, a path will be a dead end if it requires a one-syllable word to fill a four-syllable section of the input. Finally, even if there are complete false paths, the system may still get the sentence right if the correct path is found and is given a higher overall score than any incorrect path. The difference in hit and false-alarm score distributions makes this more likely. These factors, and perhaps others not yet recognized, may offset the effect of the large number of high-scoring false alarms, but speech understanding is still a difficult task, as indicated by the results of the next experiment.

E. EXPERIMENT 3 -- CONTROL STRATEGY DESIGN CHOICES

In the third experiment, the performance of the standard speech system was measured on a set of 60 test sentences, while varying four major control-strategy design choices. The sentences covered a wide range of vocabulary and included questions, commands, and elliptical sentence fragments (see Section J at the end of this chapter for a list of the test sentences). There were 10 sentences at each length of simulated speech ranging from 0.8 to 2.3 seconds at intervals of 0.3 second. The sentences averaged 5.9 words in length, with a maximum of 9 words. The choices used as experimental variables were the following:*

Island Drive or Not -- Go in both directions from arbitrary starting points in the input versus proceed strictly left-to-right from the beginning: Island driving allows the system to use words that match well anywhere in an input and to build up an interpretation around them. Left-to-right processing is simpler and less flexible but may still be more accurate and efficient than island driving.

Map All or One -- Test all the words at once at a given location versus try them one at a time and delay further testing when a good match is found: Mapping all at once lets the system know the best candidates from the acoustics and reduces the chances of following a false path. Mapping one at a time avoids exhaustive testing and will be more efficient than mapping all at once if the system does not encounter too many false alarms.

Context Checks -- Take into account the restrictions of the possible sentential contexts as part of setting priorities versus ignore the contextual restrictions except for use in eliminating already formed structures: Context checking should give more information for setting priorities and should lead to better predictions. However, the checks can be expensive and therefore may not lead to an overall improvement in performance.

* These choices are described in more detail in Chapter III, Section C.

Focus by Inhibition -- Focus the system on selected alternatives by inhibiting competition versus employ an unbiased best-first strategy: Focusing allows the system to concentrate on a particular set of potential interpretations rather than thrashing among a large number of alternatives. However, if the focus of attention is too often wrong, the net effect may be harmful to system performance.

All combinations of the four control-strategy variables were tested on the 60 sentences. This experimental design allows us to compare the 16 combinations of control choices and to evaluate, by analysis of variance, the main effects and interactions of the control strategy variables. The main effect of a variable is the change in performance it produces, averaged over all the possibilities for the other variables. The interaction of two variables tells whether the effect of one variable is the same for all possibilities of the other. The interaction of three variables tells whether the interaction of two of them is the same for all possibilities of the third, and so on. Analysis of variance is a statistical technique for computing the probability that the observed effects or interactions are really caused by the experimental variables, rather than the result of random variation (see e.g., Winer, 1971; also, see Cox, 1958, for an excellent introduction to experimental design). In other words, this method aids in evaluating results of experiments influenced by substantial random factors. In our case, the random factors include the random choices of false alarms and hit scores in simulating the mapper, and the selection of a particular sample of sentences from the much larger population of possible sentences. The statistical results for a main effect or

interaction are given in a form such as "F(1,5)=6.9, $p < .05$." This means that the F ratio (a statistic for comparing variances) for the effect or interaction has 1 and 5 degrees of freedom and has a value equal to 6.9. This in turn implies that the probability is less than .05 that the observed effect or interaction was caused by random variation alone. If the probability is given by itself in the following discussion, it is based on the these values: $F(1,5) \geq 16.3$ for $p < .01$, $F(1,5) \geq 6.6$ for $p < .05$, and $F(1,5) \geq 4.1$ for $p < .10$.

The most important performance measures for the system are accuracy (the percentage of sentences for which the correct sequence of words is found) and runtime (the computation required by the system, including simulated acoustic processing time). For these measures, the control strategy variables had large, significant effects. Before discussing the effects, we need to introduce a notation for naming the experimental designs. The capital letter "F" will refer to focus by inhibition, lower case "f" to no focus by inhibition; "C" stands for context checks, "c" for no context checks; "M" for map all at once, "m" for not map all at once; "I" for island driving, and "i" for no island driving. This notation will indicate the different combinations of choices. For example, "fCMi" refers to the system that does not use focus by inhibition, does use context checks, does map all at once, and does not island drive. Using this notation, Figure IV-5 shows the accuracy and runtime of the 16 experimental systems.

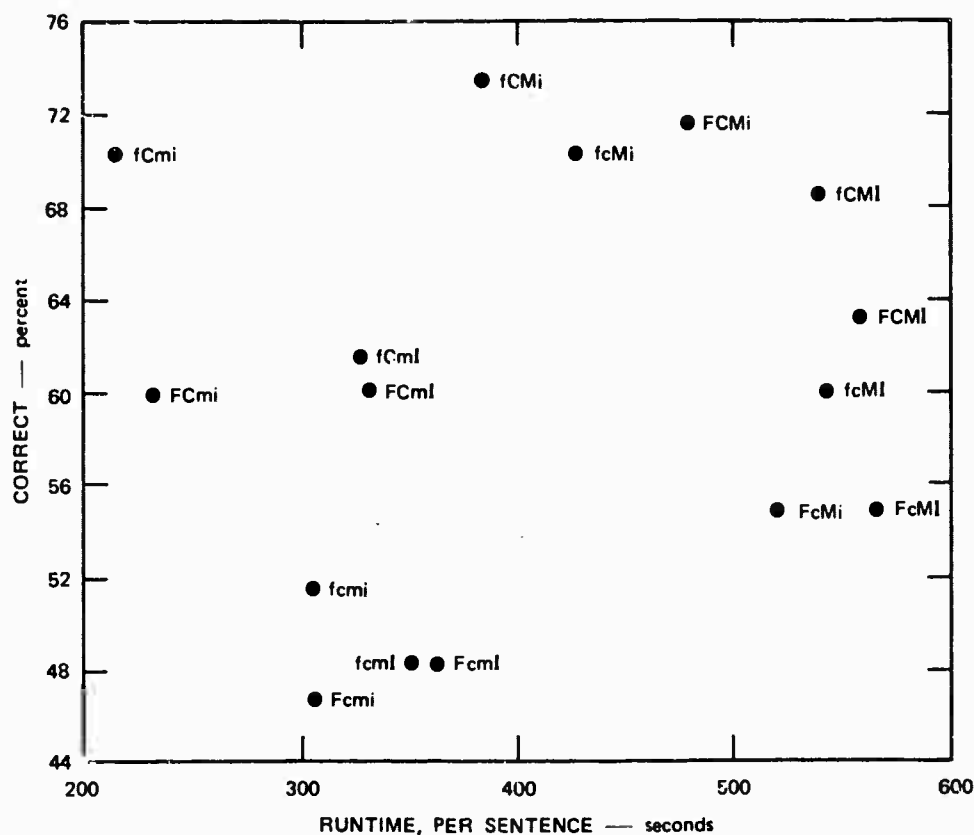


FIGURE IV-5 ACCURACY AND RUNTIME OF THE 16 DESIGNS

Notice the range of values for both measures, from 46.7% to 73.3% for accuracy, and from 221 to 559 seconds processing per sentence for runtime. These wide ranges confirm the importance of control strategy in determining system performance. With respect to the individual control variables, comparing the C-systems to the corresponding c-systems shows that context checks for priority setting result in better accuracy and faster runtimes (see Figure IV-6). Similar comparisons

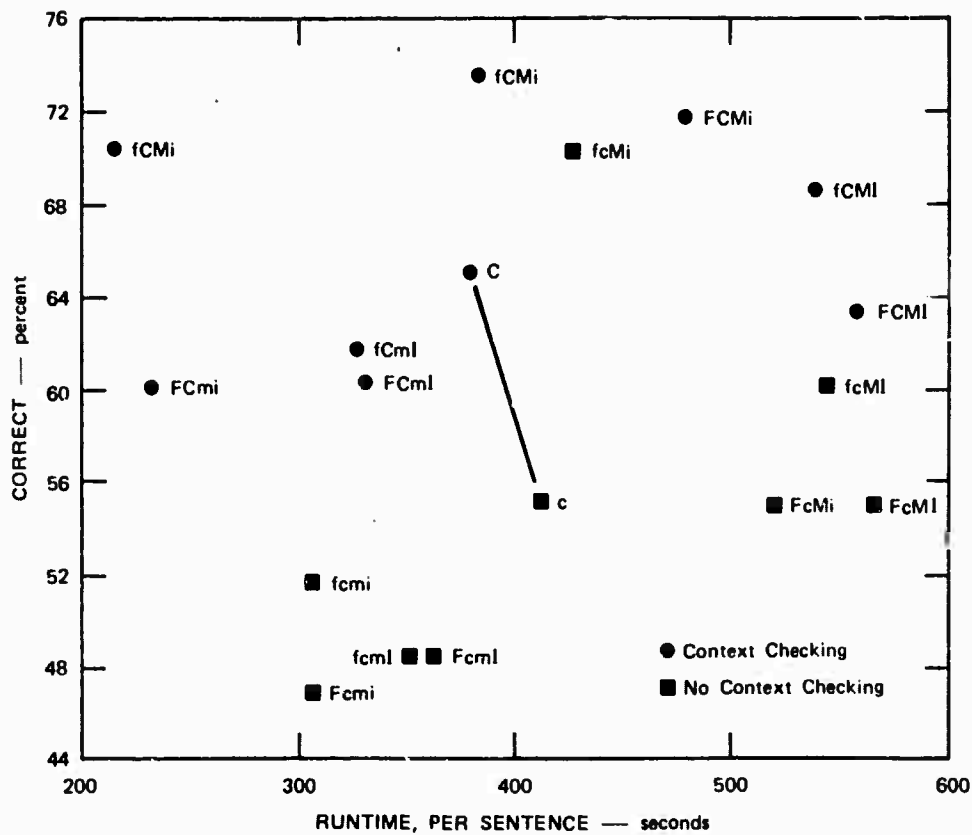


FIGURE IV-6 CONTEXT CHECKING — MAIN EFFECTS

show that mapping all at once improves accuracy but increases runtime (see Figure IV-7), while focus by inhibition and island driving both reduce the accuracy and increase the runtime (see Figure IV-8 and Figure IV-9). In the remainder of this section, we discuss these effects and propose explanations for them.

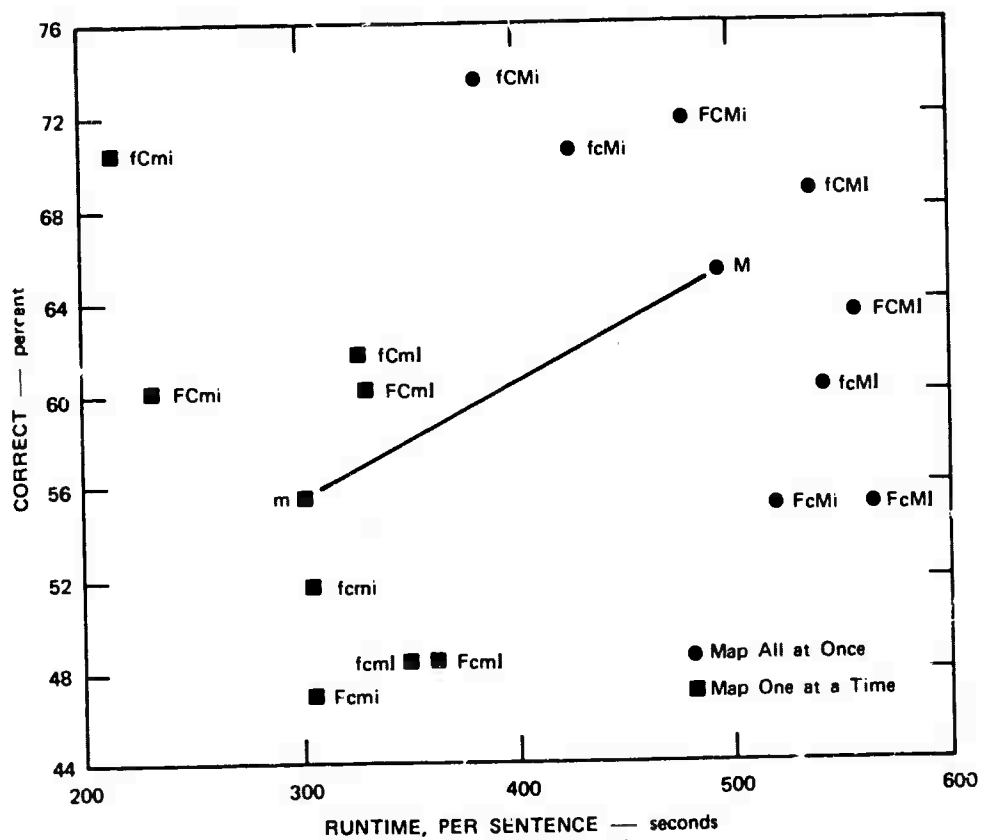


FIGURE IV-7 MAPPING ALL AT ONCE — MAIN EFFECTS

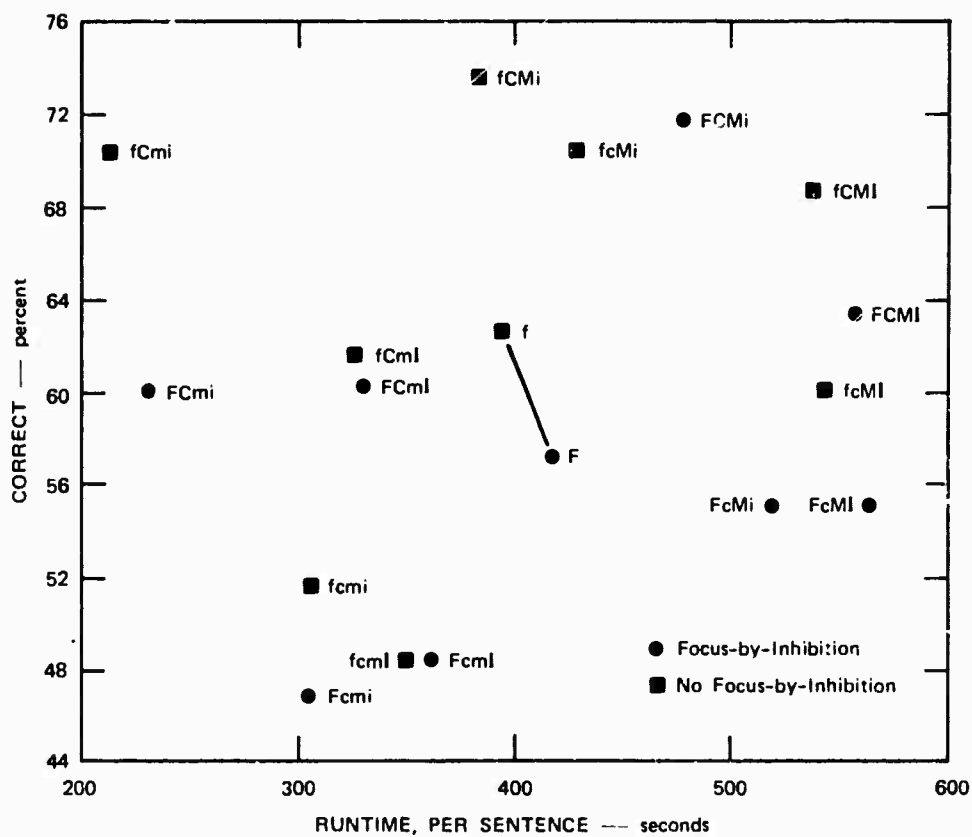


FIGURE IV-8 FOCUS-BY-INHIBITION --- MAIN EFFECTS

WITH WITHOUT DIFFERENCE

F	57.5	62.9	-5.4	*
C	66.0	54.4	11.6	*
M	64.6	55.8	8.8	*
I	58.1	62.3	-4.2	

* $p < 0.05$

Figure IV-10. MAIN EFFECTS OF VARIABLES ON PERCENT CORRECT

As previously mentioned, context checks and map all improve accuracy, while focus and island driving make it worse. The island driving effect was not significant statistically because of a large interaction with sentence length. For the long sentences, 1.7 to 2.3 seconds, island driving decreased accuracy by 15.8%, but for the short ones, 0.8 to 1.4 seconds, it actually increased accuracy by 7.5% (see Figure IV-11). There was a significant interaction ($p < 0.05$) between focus and island driving. As shown in Figure IV-12, the effect of island driving is less with focus, and the effect of focus is less with island driving. To explain this collection of results we must first consider how accuracy is influenced by control strategy.

The control strategy affects accuracy indirectly. All the strategies are 'complete' in the sense that they only reorder, and never eliminate, alternatives. If there were no false alarms, all the systems would get 100% of the test sentences correct. Even with false alarms, the strategies would get an equal percent correct, if all the possible

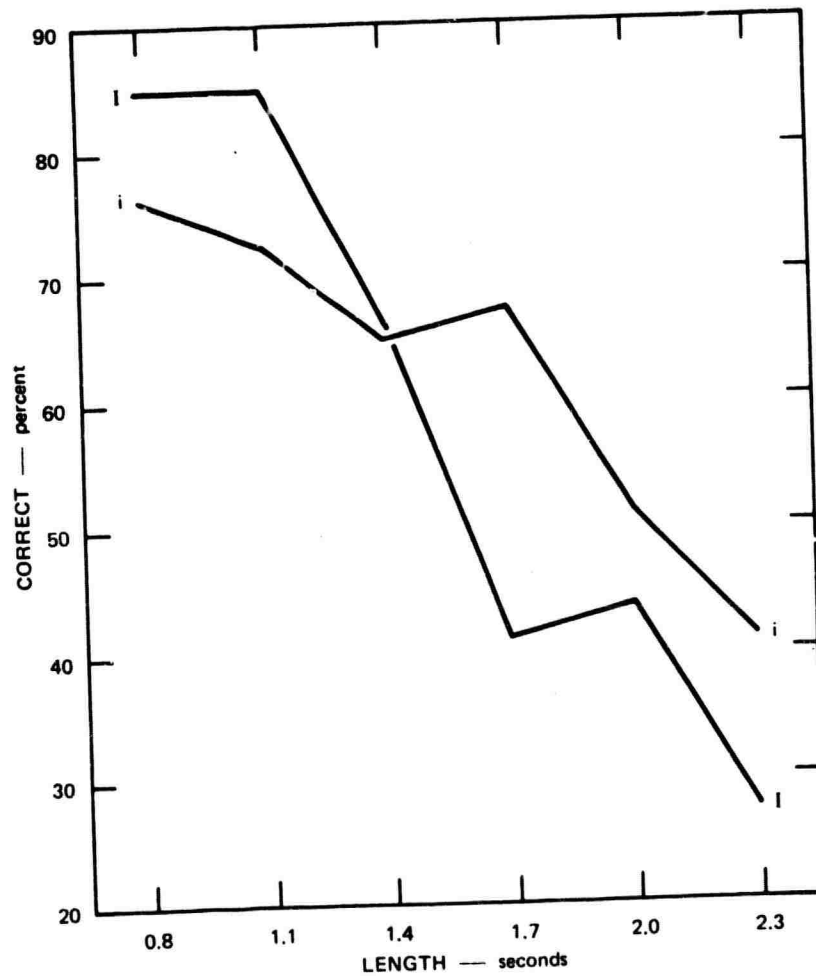


FIGURE IV-11 ACCURACY VERSUS LENGTH FOR ISLAND-DRIVING

	I	i	I-i
F	56.7	58.3	-1.6
f	59.6	66.3	-6.7
F-f	-2.9	-8.0	5.1

(percent correct)

Figure IV-12. FOCUS AND ISLAND-DRIVING INTERACTION

alternatives could be tried before the system picked an interpretation. Errors would only occur when false alarms had high enough scores to displace hits in the highest rated interpretations. However, in the actual system, the large number of alternatives makes it impossible to consider all of them in the space and time available. As a result, the order in which the alternatives are considered can affect the accuracy, and so can the demands on space and time. Control strategy thus affects accuracy indirectly by reordering alternatives and by modifying space and time requirements. To explain the accuracy effects, we must look at these other factors.

In this experiment, the storage limit was an important factor for accuracy. In the 960 tests (60 sentences times 16 systems), 578 (60.2%) were correct and 382 (39.8%) were wrong. Of the errors, 175 (46%) had an incorrect interpretation, while 207 (54%) had no interpretation at all. Since the systems could potentially get the correct answer, and no time limit was imposed until at least one interpretation had been found, all of the 207 sentences with no interpretation were a result of running out of storage.

The storage limit used in the tests was based on the number of phrases constructed. When the total reached 500, the system would stop trying new alternatives and, if any interpretation had been found, pick the highest rated interpretation as its answer. The average number of phrases constructed was 204 nonterminal and 63 terminal. The system with the best accuracy, fCM1, had the lowest average (113 nonterminals

and 45 terminals), while the system with the worst accuracy, Fcml, had one of the highest averages (260 nonterminals and 68 terminals). Overall, there was a strong negative correlation ($-.93$) between system accuracy and average number of phrases constructed (see Figure IV-13); the accuracy drops by about 1% for an increase of 6 phrase in the average storage requirements.

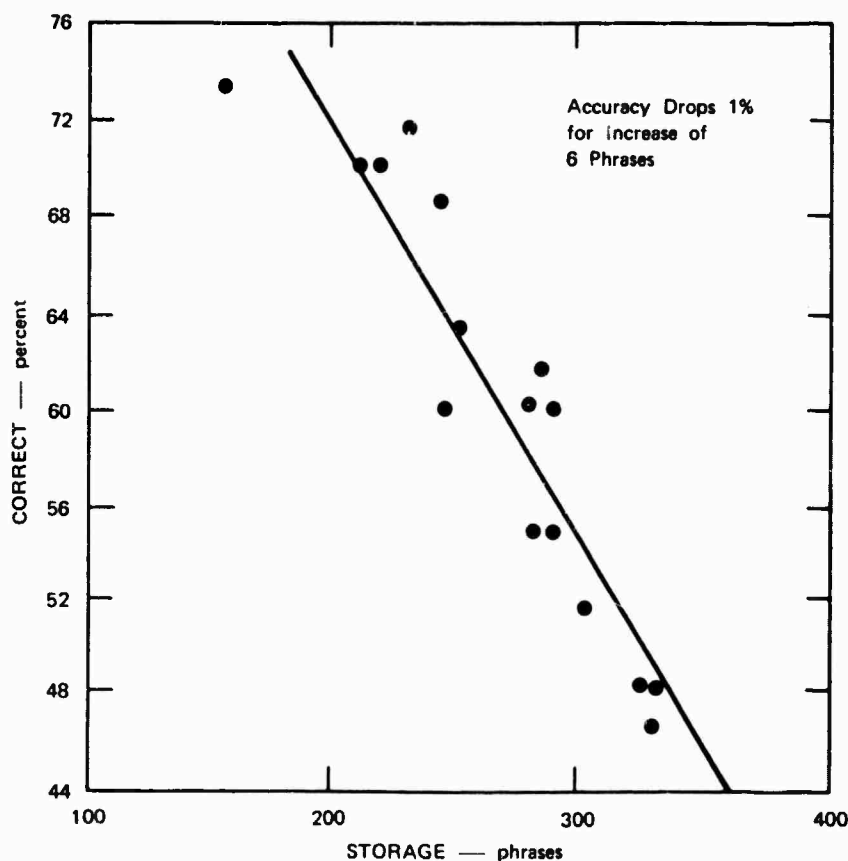


FIGURE IV-13 STORAGE AND ACCURACY FOR THE 16 SYSTEMS

Figure IV-14 shows the effects of the control variables on the number of phrases. The pattern is the same as for accuracy; context

WITH WITHOUT DIFFERENCE

F	281	253	28	*
C	240	294	-54	**
M	244	290	-46	**
I	287	247	40	

(number of phrases)

** p < .01 * p < .05

Figure IV-14. MAIN EFFECTS ON STORAGE

checks and map all have good effects, while focus and island driving have bad effects. Again, because of a large interaction with length, the island driving effect is not significant statistically. There are significant interactions, $p < 0.05$, between focus and island driving for storage, as seen in Figure IV-15, and between context checking and mapping all at once, as seen in Figure IV-16.

	I	i	I-i
F	290	272	18
f	284	222	62
F-f	6	50	-44

(number of phrases)

Figure IV-15. FOCUS AND ISLAND-DRIVING

The beneficial effects of mapping all at once are caused by a reduction in the proportion of false alarm terminal phrases. Mapping all at once significantly reduces the proportion of terminal phrases

	M	m	M-m
C	221	259	-38
c	267	322	-55
C-c	-46	-63	17

(number of phrases)

Figure IV-16. CONTEXT AND MAP-ALL INTERACTION

that are false alarms -- from 88.0% to 85.7%, $p < .01$. The false terminal proportion is in turn significantly correlated with the number of phrases (.72) and the accuracy (-.75). When the words are all mapped at once, the system is able to take advantage of the difference in false alarm and hit score distributions to reduce the likelihood of constructing false terminal phrases. Notice that a relatively small change in false terminal percentage has a large effect on system performance.

Surprisingly, context checking also results in a significant reduction in the false terminal percentage -- from 87.5% to 86.2%, $p < .01$. This reduction may be evidence that context checking is giving lower priority to looking for words adjacent to false alarms than it gives to looking next to hits. This change could affect the false terminal likelihood, since there is always a hit adjacent to a hit, while false alarms often have nothing but other false alarms next to them. In addition to its effect on false terminals, context checking may also be improving the storage requirements and accuracy by generally

improving the priority setting, thereby reducing the likelihood of following false paths.

Focus by inhibition slightly increases the proportion of false alarm terminal phrases (from 86.3% to 87.3%), but this increase is not statistically significant. The explanation of the ill effects of focus is essentially the converse of the explanation of the effects of context checking. Context checking makes performance better by improving priorities, while focus makes it worse by distorting priorities. Focus too often changes priorities to bias the system in favor of a false alarm instead of a hit. In the systems that used focus by inhibition, there was an average of 3.5 hits put in focus per sentence compared to 12.9 false alarms. Focus conflicts changed priorities in favor of a false alarm 112 times per sentence and in favor of a hit, only 15 times per sentence. Thus the priorities, and the system performance, were better with the unbiased best-first strategy than with focus by inhibition.

Island driving did not affect the false terminal proportion, but it did have bad effects on storage and accuracy for the longer sentences. To get a sentence correct, island driving must start at least one island with a hit. If all the seeds -- words selected to start islands -- are false alarms, the sentence will not be interpreted correctly. The overall average was 3.7 false alarm seeds per sentence and 0.9 hit seeds. There were one or more hit seeds in 82% of the tests using island driving. The bad effects of island driving on long

sentences was not caused by an increase in the number of false alarm seeds. The average rank of the first hit in the sequence of words for use in forming islands was 4.8, and the rank did not increase with sentence length. (The correlation between rank and length was .04). For sentences 1.7 seconds or longer, instead of an increase in the number of seeds necessary to get a hit, there was an increase in the amount of storage consumed per island. Perhaps the greater length allowed islands to grow in both directions, whereas in shorter sentences the sentence boundaries blocked one direction or the other.

The interaction of focus and island driving can be explained as the result of the storage limit. The limit put a ceiling on the size of the possible combined effect. Thus the combined effect was less than the sum of the individual effects. Similarly, the interaction between context checking and mapping all at once is a result of overlapping good effects, which consequently fail to add. The same pattern of context and map-all interaction appears in false terminal proportion, $p < .05$, and in accuracy, $F=4.00$ versus $F(1,5)=4.06$ for $p < .10$.

We now turn to a brief analysis of the sentences that got one or more interpretations but were incorrect because their highest rated interpretation was wrong. As mentioned previously, this happened in 175 tests. In 109 of these (62%), the chosen interpretation was reasonable linguistically but contained incorrect words. In 10 tests (6%), the chosen interpretation could have been eliminated by a better language definition ("Was feet one builder of the Farragut?" is an example from

these 10). Finally, 56 of the errors (32%) were harmless, in that the system would probably produce the same answer as if it had found the correct sequence of words (e.g., "What reactor does it have?" instead of "What reactors does it have?" was one of these harmless errors). If the harmless errors are counted as correct in calculating the accuracy, most accurate system, fCMI, increases in percent correct from 73.3% to 81.7%, and the average accuracy for all the systems goes up about 5.8%.

The accuracy effects have been explained in terms of storage requirements, proportions of false terminal phrases, and priorities. The important role of the storage limit raises the question of whether the accuracy effects would have disappeared if more storage had been available. We believe that the effects would have been smaller but still important. The effects on the proportion of false terminal phrases would remain, as would the presumed effects on priorities. A smaller percentage of false terminals and better priorities will cause the system to find the correct interpretation sooner, and, even if the storage limit were relaxed, the limit on runtime would remain to penalize systems that were slow to find the right answer. The effects of control strategy choices on accuracy would only vanish if space and runtime limitations were both removed, an unlikely event in view of the current performance of speech-understanding systems.

2. RUNTIME

The system runtime is another important performance measure. Here, we will use the phrase 'total runtime' to refer to the simulated acoustic processing, plus the actual processing time (on a DEC PDP KA-10) for the executive and the semantic components. The executive time is mainly spent setting priorities and parsing. The semantics time is used in constructing semantic translations and in dealing with anaphoric references and ellipsis. The reported total runtime does not include acoustic preprocessing or question answering, since neither is affected by the experimental variables.* We report results only for total, Executive, and acoustic times; semantic times are not reported, both because they are redundant given the other three measures, and because they are relatively small in comparison to the others. In analysis of variance of the runtimes, interaction with length was used as the error term.

The main effects of the control variables on total runtime are given in Figure IV-17. All the variables except context checking increase the runtime. Dividing the sentences into a short group (0.8 to 1.4 seconds) and a long group (1.7 to 2.3 seconds) shows that island driving has a much worse effect on runtime for long than for short sentences. For short sentences, island driving increased the mean

* Acoustic preprocessing takes about 160 CPU seconds per second of speech on a PDP 11/40 (according to personal communication with Iris Kameny at SDC). The time for question answering varies greatly with the complexity of the request.

WITH WITHOUT DIFFERENCE

F	417	386	31	*
C	383	421	-38	**
M	498	305	193	**
I	444	359	85	#

(seconds per sentence)

* $p < .05$ ** $p < .01$ # $p < .10$

Figure IV-17. MAIN EFFECTS ON TOTAL RUNTIME

runtime from 262 to 290 seconds, a difference of 28. For long sentences, the increase was from 457 to 598 seconds, a difference of 141. Recall that for long sentences island driving also had worse effects on accuracy and storage.

Figure IV-18 and Figure IV-19 show the main effects on executive runtime and acoustic runtime, respectively. In both cases, context checks decrease the runtime, while focus and island driving increase it. Mapping all at once improves the executive runtime but leads to a huge increase in acoustic processing time. As usual, examination of the results according to sentence length shows that island driving is worse for longer sentences. The average executive and acoustic times together account for 95% of the average total, so, as mentioned previously, we do not report separate effects for semantics.

Analysis of variance for total, executive, and acoustic runtimes reveals a significant interaction between context checking and

WITH WITHOUT DIFFERENCE

F	120	106	14	**
C	109	117	-8	#
M	90	135	-45	**
I	127	98	29	#

(seconds per sentence)

** p < .01 # p < .10

Figure IV-18. EFFECTS ON EXECUTIVE RUNTIME

WITH WITHOUT DIFFERENCE

F	276	260	16	#
C	254	282	-28	**
M	389	147	242	**
I	295	241	54	#

(seconds per sentence)

** p < .01 # p < .10

Figure IV-19. EFFECTS ON ACOUSTIC RUNTIME

mapping all at once ($p < .01$ for total and acoustics; $p < .05$ for executive). For total and acoustic runtime, the good effect of context checking was reduced when words were mapped all at once, and the increase in runtime caused by map-all was greater when also context checking. For executive runtime, both context and map-all had good effects, and there was actually a synergistic relation; context checking helped more when mapping all at once, and vice versa.

There was also a significant three-way interaction among focus, map-all, and island driving ($p < .01$ for total and acoustic runtimes; $p < .05$ for Executive). When not mapping all at once, there was negligible interaction between focus and island driving. However, when mapping all at once, the combined bad effect of focus and island driving was significantly less than the sum of their individual bad effects.

The runtime results follow basically the same pattern as the accuracy and storage results. Focus and island driving have bad effects, with worse results from island driving for longer sentences, while context checking has consistently good effects. Map-all has a good effect on Executive runtime, but, unfortunately, it causes large increases in acoustic and total runtimes. The only inconsistency with the previous pattern of effects for accuracy and storage is the bad effect of map-all on the acoustic runtime. This fact is explained by pointing out that the mapper was designed for mapping words one at a time and, in the simulation, does not accumulate or share information to make subsequent tests more efficient. Finally, it is noteworthy that the extra effort for context checking resulted in a net decrease in processing time. For example, fCMI spent an average of 6.3 seconds more per sentence doing extra processing for context checks, but it was still 41 seconds per sentence faster than fcMI.

The runtime figures above are in units of seconds used to process a sentence. A more common unit for runtime is seconds per

second of speech. This is a reasonable scale if the runtime can be approximated by a linear function of sentence length with a zero intercept. Both assumptions, linearity and zero intercept, are consistent with our data. No significant nonlinearity was found by an F test of the variance of the mean for each length about the regression line, relative to the combined variance of the sentences within a given length (for instance, the data for fCm1 gave $F=1.37$ versus $F(4,54)=1.41$ for $p < .25$). Moreover, the 95% confidence interval for the intercept of the regression line included the origin. With this justification, we used zero-intercept linear regression to calculate the processing times per second of speech and their 95% confidence intervals.

The results for the fastest system, fCm1, were 141, plus or minus 14 seconds processing per second of speech for total runtime; 66, plus or minus 9 for executive runtime; and 63, plus or minus 7 for acoustic runtime. The results for the most accurate system, fCM1, were 247, plus or minus 21 for total runtime; 34, plus or minus 6 for executive runtime; and 205, plus or minus 14 for acoustic runtime. Thus, for fCM1, 83% of the total runtime slope comes from acoustic processing, 14% from the executive, and the remaining 3% from the semantics. Clearly, the best approach to improving fCM1 runtime is to redesign the mapper for mapping all at once. The potential is large for sharing work to improve efficiency in the mapper, since the data show that fCM1 is mapping all the words at an average of 13 out of the 20 possible positions per second of speech.

3. REVIEW OF THE EFFECTS

The only control strategy choice with mixed effects in Experiment 3 is whether or not to map all at once. Mapping all at once improves accuracy and executive runtime, but at a large cost in acoustic and total runtime. Redesign of the mapper, perhaps along the lines of the BBN lexical retrieval component (see Klovstad in Woods et al., 1976b), could undoubtedly resolve this choice in favor of mapping all at once. For example, just cutting the acoustic processing in half would make the fCmi system about as fast as the fCmi system. The choice, whether to map all or not, is explored further in Experiment 5.

The overall effects of island driving were bad, and they were particularly bad for longer sentences. Island driving was hurt by false alarm seeds, especially when the sentence was long enough for the islands to grow in both directions. Perhaps island driving can be modified to overcome this problem. For example, a multiword seed technique like Hearsay-II's (see Chapter III, Section E.2) might reduce the number of false alarm seeds, and a restriction like the one used by BBN to keep seeds near the start of the utterance might reduce the storage needed per island. Another alternative would be to pick seeds anywhere in the utterance but to restrict them to growing in one direction to an utterance boundary before allowing them to grow in the other direction. Island driving did give higher accuracy for the shorter sentences and correctly answered some of the sentences that fCmi missed (see Section H), so further effort is justified to look for

versions of island driving that have good effects for long sentences as well as for short ones.

The effects of focus by inhibition were bad on all measures. The cause of the bad effects was too much focusing on false alarms, so we have tried a modified version that is much more conservative about which words to put in focus. It uses the false-alarm likelihood estimates as a primary criterion in selecting words for focus. The modified focus method was tested on the 60 utterances using the FCMi system, which was the best of the original focus systems. The results are shown in Figure IV-20. The modification greatly reduced the number of false alarms in focus and improved the FCMi performance of all measures. In fact, the modified-FCMi is the most accurate of all the systems tested; it correctly answered all the sentences that FCMi did, plus one more. However, it is still slightly worse than FCMi in storage and runtime. The differences between FCMi and the modified-FCMi are small (because so few words are put in focus by the modified technique), but they suggest that focus by inhibition might have significant good effects if further effort was devoted to tuning the algorithm for selecting focus words.

Context checking had uniformly good effects. For both accuracy and runtime, it was worth the extra effort to get better priority setting. This result clearly depends on the fact that we put a large amount of the system's knowledge into the rule procedures of the language definition rather than into the structural declarations. It

	new-FCMi	old-FCMi	FCMi
Words in Focus			
Hits	1.3	3.9	0.0
False Alarms	1.6	8.2	0.0
Focus Conflicts			
Hits	4.0	13.3	0.0
False Alarms	12.8	75.8	0.0
Raw Accuracy, %	75.0	71.7	73.3
Forgiving, %	81.7	76.7	81.7
Runtime (sec/sent)	392	477	385
Executive "	60	83	53
Acoustic "	321	377	320
Storage (phrases)	165	231	158

Figure IV-20. EFFECTS OF MODIFIED FOCUS BY INHIBITION

would be interesting to repeat these tests with different language definitions that had the same linguistic scope but put more information into the structure and less in the procedures.

F. EXPERIMENT 4 -- GAPS AND OVERLAPS

The data from Experiment 1 do not aid us in simulating the mapper's performance when called on to test whether two words it has accepted individually are also acceptable as a contiguous pair. Such tests, referred to as 'phrase mapping', are necessary whenever words and phrases are combined to form larger units. In the experimental simulation of the mapper, we replaced phrase mapping by a simple threshold test; we allowed gaps and overlaps of up to 0.05 second of speech, but rejected those that were larger. Experiment 4 tests the effect of different values of the gap-overlap parameter on the

	GAP-OVERLAP SIZE		
	0.00	0.05	0.10
Fanout with acoustics (words)	6.6	18.0	29.4
Rank of hit in fanout	1.9	3.7	5.6
Raw accuracy, %	96.7	73.3	48.3
Forgiving accuracy, %	98.3	81.7	58.3
False terminal, %	58.2	83.2	89.1
Number of nonterminals	31	113	217
Total runtime (sec/sec-speech)	140	247	333
Executive runtime "	10	34	69
Acoustic runtime "	128	205	243

Figure IV-21. EFFECTS OF GAP-OVERLAP PARAMETER

performance of the fCMI system from Experiment 3. Figure IV-21 gives the results for a variety of measures with gap-overlap sizes of 0.00, 0.05, and 0.10 second.

The performance is much better for 0.00 and much worse for 0.10 second of gap or overlap. The observed distribution of gaps and overlaps is shown in Figure IV-22. Notice that a technique using a simple threshold on the size of gaps and overlaps would not be acceptable in practice; the threshold would have to be at least 15 centiseconds, and the data reported in Figure IV-21 suggest that the resulting performance would be terrible. This is strong evidence for the importance of special acoustic tests to verify word-pair junctions. Such tests can lead to a large reduction in the average hit rank and, consequently, to significant improvements in both accuracy and runtime.

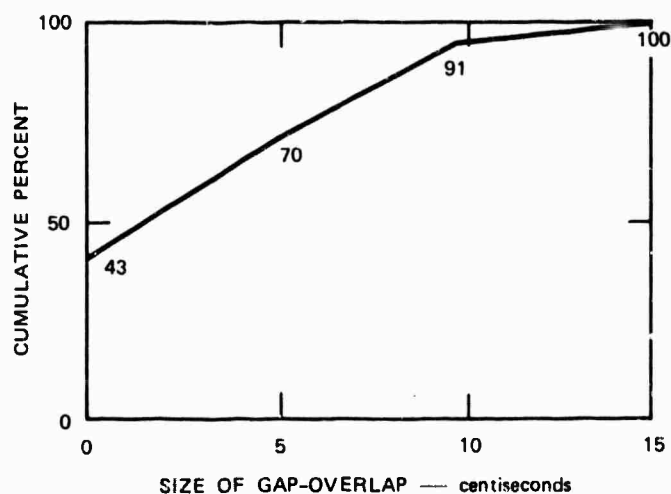


FIGURE IV-22 OBSERVED DISTRIBUTION OF GAPS AND OVERLAPS

G. EXPERIMENT 5 -- INCREASED VOCABULARY AND IMPROVED ACOUSTICS

Experiment 5 studies the effects of increased vocabulary size and improved acoustic-processing accuracy. As test systems, we use fCm1 and fCm1 from Experiment 3. These are the best systems for accuracy and speed, respectively, and they also give us more information about the map-all control strategy choice. Thus there are three experimental variables: vocabulary size, acoustic accuracy, and map-all. Data for two of the eight combinations, map-all or not for smaller vocabulary and regular acoustic accuracy, come from Experiment 3. For Experiment 5, the other six combinations were tested to provide a complete set of data for analysis of the effects of the variables.

The large vocabulary is a 451-word superset of the 305-word vocabulary used in the other experiments. The data gathered in Experiment 1 showed that, with the 451-word vocabulary, the mapper made 2026 false alarms and had a false alarm rate of 142 false alarms per second of speech (compared with 114 for the 305-word vocabulary). Using the Experiment 1 information, the mapper performance was simulated for the large vocabulary on the same set of 60 test sentences.

Improved acoustic-processing accuracy was simulated by a 7% downward stretch of the false alarm score distribution, while leaving the hit scores unchanged. In other words, a false alarm score X , in the range 45 to 100, was replaced by $1.07X - 7$. If the result was below the threshold of 45, the false alarm was eliminated. This process reduced the number of false alarms for the 305-word vocabulary from 1564 to 1204, and for the 451-word vocabulary, from 2026 to 1541. Because the subthreshold scores were eliminated, the simulated improvement left the average false alarm score almost unchanged: for the 305-word vocabulary, it went from 59.4 to 60.2, and for the 451-word vocabulary, it went from 58.2 to 58.8. We feel that an improvement in acoustic accuracy of the magnitude simulated here could have been achieved by careful tuning of the mapper.

Figure IV-23 records the accuracy results using the notation "M" for tests with mapping all at once, "m" for those without, "A" for systems with improved acoustic accuracy, "a" for those without, "V" for systems with increased vocabulary, and "v" for those without. Improved

	AMv	Amv	aMv	AMV	amv	AmV	aMV	amV
Raw, %	85.0	78.3	73.3	71.6	70.0	68.3	68.3	53.3
Forgiving, %	95.0	85.0	81.7	78.3	76.7	76.7	75.0	53.3

Figure IV-23. ACCURACY RESULTS

acoustics raises fCMI accuracy from 73.3% to 85.0%, or from 81.7% to 95.0% if harmless errors are forgiven. However, if vocabulary size is also increased, accuracy drops slightly from 73.3% to 71.6%. Thus, in this experiment, a 7% improvement in acoustic accuracy almost compensates for a 48% increase in vocabulary. Comparison of the M-results to the m-results shows that map-all consistently helps accuracy.

The main effects on accuracy and several other measures are given in Figure IV-24. Improved acoustics leads to big gains in accuracy, storage, and runtime. Increased vocabulary makes performance worse, but at least the system does not collapse. As in Experiment 3, mapping all at once improves everything except acoustic and total runtimes.

There were few significant interactions. Vocabulary size and mapping all at once interacted significantly for acoustic runtime ($p < .05$) and for total runtime ($p < .10$). Figure IV-25 shows that the increase caused by map-all is greater for the bigger vocabulary, and, surprisingly, that the increase in vocabulary size leads to a reduction in processing, if the system is not mapping all at once.

WITH WITHOUT DIFFERENCE

Raw Accuracy (percent)				
A	75.8	66.3	9.5	**
V	65.4	73.7	-11.3	#
M	74.6	67.5	7.1	*
Phrases (total number terminal and nonterminal)				
A	155	208	-53	**
V	204	159	45	**
M	156	206	-51	**
False Terminals (percent)				
A	80.6	85.9	-5.3	**
V	84.3	82.1	2.2	
M	81.7	84.8	-3.1	**
Total Runtime (seconds/sentence)				
A	266	320	-54	**
V	312	275	37	*
M	383	204	179	**
Acoustic Runtime (seconds/sentence)				
A	187	213	-26	**
V	205	195	10	
M	315	84	231	**
Executive Runtime (seconds/sentence)				
A	66	89	-23	**
V	88	67	21	**
M	55	101	-46	**

** p < .01 * p < .05 # p < .10

Figure IV-24. MAIN EFFECTS OF ACOUSTICS, VOCABULARY, AND MAP-ALL

	M	m	M-m
V	335	75	260
v	296	94	202
V-v	39	-19	58

(seconds/sentence)

Figure IV-25. VOCABULARY AND MAP-ALL INTERACTION FOR ACOUSTIC RUNTIME

Mapping all at once also interacted significantly with acoustics for acoustic runtime ($p < .01$), total runtime ($p < .01$), and false terminal percentage ($p < .05$). All cases were similar to the one shown in Figure IV-26. There was a synergistic interaction causing mapping all at once to be more effective with better acoustics, and vice versa. This result is readily explained since map-all is designed to take advantage of the difference between false-alarm and hit-score distributions, and improving the acoustics enhances that difference by reducing the number of high scoring false alarms.

	M	m	M-m
A	78.6	82.6	-4.0
a	84.8	87.0	-2.2
A-a	-6.2	-4.4	-1.8

(percent)

Figure IV-26. ACOUSTICS AND MAP-ALL INTERACTION FOR FALSE TERMINALS

In addition to the main tests for Experiment 5, we also ran another test to study the effect of improved acoustics on a system using island driving. The best island driving system from Experiment 3 was fCMI. When tested on the 305-word vocabulary with 7% simulated improvement in acoustics, fCMI gained in accuracy from 68.3% to 78.3%. It was still below the non-island driving fCMI, and the gap between them remained large. (Recall that fCMI went from 73.3% to 85.0%.) Thus, improvements in acoustics alone appear unlikely to be able to solve the problems with this version of island driving.

In summary, this experiment has given us information about how badly the system is hurt by increased vocabulary, and how much it is helped by improved acoustics. With respect to the control-strategy design choices, further evidence appeared in favor of mapping all at once, and against the current version of island driving.

H. DETAILED MEASUREMENTS OF EXECUTIVE OPERATION

This section gives a detailed breakdown of the statistics for the most accurate of the Experiment 3 systems, fCM1. Based on the performance for fCM1 on the 60 test sentences in Experiment 3, we report information regarding the composition of the parse net, the effects of lookahead, the performance of the context-checking procedures, the processing time for the major Executive procedures, and the breakdown of the accuracy results according to the existence and relative scores of correct and incorrect interpretations. Because of its level of detail, this section presupposes familiarity with the description of the Executive given in Section D of Chapter III.

Figure IV-27 shows the average composition of the parse net at the end of processing an utterance. Notice that there are more consumer-to-prediction links than there are predictions (78 versus 61), so there is some sharing taking place. To estimate the amount of sharing, we computed what the total size would be if the parse net were a tree instead of a network and all of the shared structures were

133	Nonterminal phrases per sentence 38 complete, 50 partial, 25 empty
45	Terminal phrases per sentence
61	Predictions per sentence
78	Consumer-to-prediction links per sentence

Figure IV-27. COMPOSITION OF THE PARSE NET

duplicated.* The average number of phrases plus predictions was 220 in the actual parse net; expanding the net into a tree increased the average to 404, an 83% increase. Thus, while only 17 out of the 78 consumer-links (22% of them) went to another consumer's prediction, the overall savings were quite large.

Whereas Figure IV-27 shows the number of phrases and predictions that were actually constructed, Figure IV-28 shows how many were blocked for various reasons. There were 42.3 phrases per sentence rejected by language factor statements, and of these, syntactic factors, which are usually tested first because they are less expensive computationally, accounted for over 90%. The preliminary tests in the add-constituent procedure (times, phrase mapping, and lookahead) blocked 5.1 phrases per sentence. In 17.8 cases per sentence, the same terminal or nonterminal phrase already existed, so the construction of a

 * The fCM1 system works left-to-right and there is no left-recursion in the rules of the language, so the parse net does not have loops and can be converted to a finite tree.

42.3	Factor Rejections	
38.3	(90.3%) by syntactic factors	
1.5	(3.5%) by case-grammar factors	
0.3	(0.7%) by semantic translation factors	
2.3	(5.4%) by discourse factors	
5.1	Add-Constituent Preliminary Tests	
3.1	in Part 1	
2.0	in Part 2	
17.8	Same Phrase Already Existed	
4.5	Nonterminal	
13.3	Terminal	
31.4	Phrases and Predictions Blocked by Lookahead in the Predict Task	

96.6	Total	
	(per sentence)	

Figure IV-28. BLOCKING OF PHRASES AND PREDICTIONS

duplicate was blocked.* The final type of blocking is lookahead in the predict task, which accounted for 31.4 blocked phrases and predictions per sentence.

The data in Figure IV-28 show that the lookahead mechanism is providing a substantial constraint, so it is of interest to compare the performance of the system with and without lookahead (see Figure IV-29). Lookahead has good effects on accuracy, storage, and Executive runtime, but not on acoustic runtime or total runtime. The

* It is possible to have such duplication even with a left-to-right control strategy because of the looseness in the time constraints. For example, a word starting at position 45 can be found by predictions 40, 45, and 50, leading to two blocked duplicates.

	WITH	WITHOUT	DIFFERENCE
Raw Accuracy, %	73.3	71.7	1.6
Forgiving, %	81.7	76.7	5.0
Total Runtime (sec/sent)	385	287	98
Exec Runtime (sec/sent)	53	64	-11
Acoustic Runtime (sec/sent)	320	208	112
Storage (phrases)	158	192	-34

Figure IV-29. EFFECTS OF LOOKAHEAD

bad effects result from an increase in the number of places per sentence where words were tested -- up from 13 without lookahead, to 20 with. Lookahead is causing the system to 'peek' at places that without lookahead it would simply ignore. The extra testing done with lookahead would be less expensive with a redesigned mapper, but the cost undoubtedly would not decrease enough to compensate for the existing difference.* Lookahead appears to help accuracy somewhat, so rather than simply discard it, we feel that further effort is called for to design a version of the system that uses lookahead in a more efficient way.

Unlike lookahead, context checking has uniformly good effects. As mentioned previously, an average of about 6.3 seconds of processing per sentence was spent doing context checking, and this effort resulted in a net decrease of 41 seconds per sentence in the total runtime (based on comparison with the results for the best system without context-

 * Based on the data in Figure IV-29 the acoustic runtime would have to drop to one-eighth of its current level to cause the total runtimes to be the same (subtract seven-eighths of the acoustic runtime from the total runtime: with lookahead, $385 - 320 * 7/8 = 105$; without lookahead, $287 - 208 * 7/8 = 105$).

checking (fcM1) -- the savings were 19.4 seconds in Executive processing, 17.3 seconds in acoustics, and 4.2 in semantics). There was an average of 50 rating assignments made by context checking per sentence, with the construction of 78 virtual phrases and 28 complete consumer paths (average length of a complete path, 1.5 virtual phrases). The rejection of a virtual phrase by rule factors caused 35 paths to be terminated per sentence. Ten paths per sentence were blocked by the heuristic search procedure because their priority was less than the established lower bound. Note that the number of rating assignments by context checking equals the number of partial nonterminal phrases (given in Figure IV-27). Thus, there was no recalculation of ratings for partial nonterminal phrases after the first assignment. Also, since there were only 28 complete paths, at most 28 of the partial nonterminals were allowed to make predictions.* The ones without a complete consumer path received a rating of zero and were not added to predict-sets. This result helps to explain the value of context checking; about half of the partial nonterminals that were all right with respect to local tests and fit their consumers' structural requirements were rejected by their consumers' factor statements and, therefore, were given a zero rating.

The processing time for context checking appears to be well spent, but how was the time spent for the rest of the Executive procedures? Figure IV-30 shows the total time per sentence, the number of calls

* It may have been fewer than 28 if some rating assignments created more than one complete path.

FUNCTION	TOTAL TIME (seconds)	CALLS	PER CALL (milliseconds)	PERCENT OF EXECUTIVE
Word Task				
Create-word-set	6.1	48	127	10.8
Get-a-word	1.5	18	83	2.7
Create-terminal	3.9	70	56	6.9
Distribute-phrase	1.5	65	23	2.7
Add-constituent				
Top-level	2.5	112	22	4.4
Prelim-part-1	1.5	112	13	2.7
Prelim-part-2	4.2	109	39	7.5
Complete-phrase	6.0	107	56	10.7
Rule procedure	3.2	41	78	5.7
Consumer-checks	3.3	156	21	5.9
Incomplete-phrase	3.8	71	54	6.7
Rule procedure	1.7	71	24	3.0
Add-predict-sets	1.1	35	31	2.0
Predict-task				
Create-Subnet	5.3	19	279	9.4
Assign-Ratings	1.6	19	84	2.8
Cleanup	0.5	19	26	0.9
Context-checking				
Virtual-phrase	3.4	78	44	6.0
Rule procedure	1.9	78	24	3.4
Search	1.7	50	34	3.0
Top level	1.6	1	1584	2.8
(per sentence)				

Figure IV-30. TIMING BREAKDOWN

per sentence, the time per call, and the percentage of Executive runtime, for the major Executive routines. Notice that the combined rule procedure execution time for complete, incomplete, and virtual phrases is only 12.1% of the Executive processing. The time for the rule procedure with a complete phrase is about 78 milliseconds, while with an incomplete or virtual phrase the time is only 24 milliseconds. Given the size of the rule procedures and the capabilities of the

INTERLISP compiler and the PDP KA-10, these times probably cannot be improved significantly. The create-subnet procedure stands out as taking the most time per call, 279 milliseconds on the average. However, it is a complex operation and that amount of time does not seem unreasonable for it. Perhaps the main conclusion to be drawn from the timing data in Figure IV-30 is that big improvements in processing time will not come from discovering and correcting implementation blunders in the Executive -- instead, nontrivial design innovations will be needed.

The final measurements to be discussed deal with the accuracy of the fCMi system. Figure IV-31 shows the accuracy breakdown in terms of the existence and relative scores of correct and incorrect interpretations. Overall, fCMi got 44 sentences correct and missed 16. Of the 16 errors, five were 'harmless'. Three of the harmless errors consisted of leaving out a plural morpheme. These accounted for all of the cases in which an interpretation was found but had a worse score than an incorrect interpretation that was also found. The other two harmless errors were among the cases in which only incorrect interpretations were found. In one, the system picked an interpretation containing "has" instead of a plural morpheme followed by "have". In the other, a singular verb suffix was accepted instead of a past tense suffix. In both cases, the incorrect interpretation had a higher score than would have been given to the correct interpretation (if it had been found). Thus, because of high scoring false alarms, the optimal solution (the interpretation with the highest score possible) was not the correct solution.

32 times only got correct interpretation
8 times only got incorrect interpretation
3 times got no interpretation
12 times correct score better than bad score
2 times correct score same as bad score
3 times correct score worse than bad score

60 total -- 44 correct and 16 errors.

Figure IV-31. ACCURACY BREAKDOWN

The two cases having correct and incorrect interpretations with equal scores were caused by the presence of false alarms that could not be rejected by linguistic considerations alone. The three cases getting no interpretation all had a low scoring word in either the first or second position (mapper scores of 59 or less), and in two of the cases, island driving (by the fCMI system) succeeded in finding the correct answer.

The eight cases in which only an incorrect interpretation was found can be divided into three categories: forgiven errors, optimal but not correct, and suboptimal. As mentioned previously, two of the eight with no correct interpretation were forgiven errors. Three were the result of finding an optimal interpretation that was not correct. [Surprisingly, island driving (fCMI) got one of these correct by stopping with a suboptimal, correct interpretation.] The final three were the result of stopping with a suboptimal, incorrect interpretation. In these last three, the correct interpretation started with either a bad score (56) or a small word ("how" or "the"). In each case, island driving (fCMI) got the correct answer.

Of the 16 sentences that fCMI missed, five were forgivable acoustic errors, six were correctly interpreted by an island driving system (fCMI), four were the result of finding optimal but incorrect interpretations, and one had so many attractive false paths that it could not be handled within the storage limits by any of the systems. These results indicate that a different control-strategy might have correctly answered at least five sentences more than fCMI did: three for which fCMI picked a suboptimal interpretation and two for which fCMI found no interpretation although fCMI found the correct one. Such an improved strategy would have an 81.7% accuracy (90.0% forgiving), with 'nonforgiven' errors traceable to either acoustics (five cases) or storage limits (one case). This result gives a rough upper bound for improvements by modifying the control strategy versus modifying the acoustics. Of the 16 errors by fCMI, five were the result of the control-strategy failing to find the optimal interpretation, and 11 were the result of acoustic errors -- but five of the 11 acoustic errors could be forgiven.

I. CONCLUSION

Reviewing the series of experiments, the first experiment showed that the acoustic processing component called the 'mapper' had a high false-alarm rate, but tended to give better scores to hits than to false alarms. In the second experiment, we measured the number of alternatives open to the system for extending segments of sentences. The size of the fanout helps to explain the difficulty of speech understanding. The third experiment studied the effects on system performance of four control-strategy design choices. Focus by inhibition and island driving had bad effects, while context checks for priority setting had good effects. Mapping all at once had good effects on everything except acoustic and total runtime, and these bad effects could probably be eliminated by redesign of the mapper. The fourth experiment varied the size of allowed gaps and overlaps between words and showed the potential value of special acoustics tests to verify word-pair junctions. The fifth experiment gave quantitative measures of how badly the system is hurt by increased vocabulary, and how much it is helped by improved acoustic accuracy. The experiment also provided more information about the control choices. The final study considered detailed measurements of the Executive performance and provided insights into the use of time and storage and the kinds of errors made by the system.

Overall, the series of experiments gives a better understanding of the system performance and suggests new possibilities for further

research. With respect to methodology for analyzing complex systems, the results indicate that experimentation using analysis of variance is a useful technique in computer science (as suggested by Newell, 1975). It is a technique that has been widely used in other areas of science and technology, but it has seen almost no use in computer science.* Our experience shows that analysis of variance and related statistical methods can provide a productive paradigm for the study of complex computer systems.

J. TEST SENTENCES

The 60 sentences listed below were used in the control strategy experiments. The sentences are grouped according to their simulated length in seconds of speech. Processing for all of the sentences assumed a dialog context in which the preceding utterance was "What is the speed of the Batfish?". For example, the test utterance "Submerged displacement?" was interpreted by the system as meaning "What is the submerged displacement of the Batfish?".

2.3 seconds

Is the size of the Hammerhead 2000 tons?
Was Portsmouth Naval Shipyard the builder of the Seadragon?
Which subs have a length of 300 feet?
How many subs did Puget Sound Naval Yard manufacture?
What engine was manufactured by General Dynamics?
Whose ships did the Electric Boat Company construct?
Which destroyers were constructed by Cammell Laird Company?
Which AGFF did H. M. Dockyard construct?

* However, Gillogy (Carnegie-Mellon University, Computer Science Department Ph.D. thesis, in progress) applies analysis of variance to the study of a chess program.

How many diesel frigates does Great Britain have?
Did Bethlehem Steel Company manufacture a cruiser?

2.0 seconds

Were the Lafayettes built by Todd Pacific Shipyards?
Which countries have conventional submarines?
Which frigates were built by Newport News Shipyard?
Does the Swordfish have a speed of 30 knots?
What is the surface displacement of the Queenfish?
What categories of submarines are there?
Did Vickers Armstrongs Limited construct the Olympus?
Does the United States own that submarine?
What standard displacement does the Seahorse have?
What training submarines does England own?

1.7 seconds

Who constructed the English cruisers?
How many frigates are owned by the U.S.?
How many cruisers does England own?
How many classes of subs are there?
Do Resolutions have two reactors?
Is Britain the owner of the Conqueror?
Is the Renown a British submarine?
How many patrol submarines are there?
How many countries have CGNs?
Name the owners of aircraft carriers.

1.4 seconds

What country owns the Superb?
Who was the builder of the Jack?
Was its builder Avondale Shipyards?
Do we have ten diesel carriers?
How fast are the Graybacks?
What reactors does it have?
Was it built by Norfolk Navy Yard?
How many turbines do Brookes have?
Which nuclear carriers do we own?
Print the draft of the Scamp.

1.1 seconds

List the CHGs.
How many Darters are there?
Is it a research sub?
Who is the owner of it?
The speed of the Onslaught?
How fast is the Trout?
How many CGs are there?

Speed of the Bluefish?
What engines are there?
Is a CV a submarine?

0.8 seconds

Submerged displacement?
How long is it?
Whose ship is it?
Is it owned by us?
The Constellation?
Its surface speed?
Who constructed it?
What is its size?
What displacement?
Displacement of it?

V THE REPRESENTATION OF SEMANTIC KNOWLEDGE

Prepared by Gary G. Hendrix

CONTENTS:

- A. Introduction
- B. The Role of Semantic Representation
- C. Basic Network Notions
 - 1. A Preliminary Example
 - 2. Restrictions on Nodes and Arcs
 - 3. The Hierarchical Taxonomy
- D. Partitioning
 - 1. Spaces
 - 2. Vistas
 - 3. Supernodes
- E. Higher-Order Structures
 - 1. Logical Connectives
 - a. Conjunction
 - b. Disjunction
 - c. Negation
 - d. Implication
 - e. Spaces as Conjunctions
 - 2. Quantification
 - a. The Implicit Existentials of Propositions
 - b. The Orthogonal Partitioning Approach to Quantification
 - c. The Implicit Existential Approach to Quantification
 - d. Streamlining the Implicit Existential Approach
 - e. Examples of the Overlap Shorthand
 - f. Quantification in the Hierarchical Taxonomy
 - g. Delineations
 - 3. Other Higher-Order Structures
 - a. Representing Yes/No Queries
 - b. Representing WH Queries
 - c. Representing How-Many Queries
- F. Augmentations
 - 1. Property Lists
 - 2. Procedural Augmentation
- G. Supports for Diverse Tasks
 - 1. Focus
 - 2. Scratch Spaces
 - 3. Relating Syntax to Semantics
- H. Linearized Net Notation
- I. Applying the Representation

A. INTRODUCTION

This chapter describes the framework for knowledge representation that underlies those semantic-oriented components of the SRI speech understanding system that deal with the content of communication as opposed to (or in addition to) its surface form. The representation scheme embodies the system's knowledge about the nature of the task domain (that portion of the outside world with which the system is conversant) and serves as the medium for recording and communicating semantic information among the relevant system components during the interpretation of an utterance. The components that make use of the representation scheme and encoded knowledge include the semantic composition routines, the discourse component, the deduction component, and the English generator. The development of the representation has been strongly influenced by the requirements of the components; correspondingly, the representation has helped, at a fundamental level, to shape the design of all components dealing with content and plays a central role in their coordination.*

The representation scheme builds upon and extends the notion of semantic networks as described by Simmons (1973), Shapiro (1971), Rumelhart and Norman (1972), and Schank (1973). However, the network structure used in the SRI speech understanding system differs from that of other nets in that nodes and arcs are partitioned into 'spaces'

* The most recent editions of the computer primitives for constructing and manipulating procedurally augmented partitioned semantic networks were programmed by Jonathan Slocum and Ann Robinson.

(Hendrix, 1975a,b). These spaces, playing a role in networks roughly analogous to that played by parentheses in strings and lists, group information into bundles that help to condense and organize the network's encoding of information. In particular, partitioning is used in the speech understanding system:

- * To encode logical connectives and higher-order predicates, especially quantifiers.
- * To encode the association between surface and deep structure. Each syntactic unit of an input utterance is cross-indexed with its translation image in the network.
- * To interrelate new inputs with previous network knowledge while maintaining a definite boundary between the new and the old.
- * To simultaneously encode in one network structure the multiple hypotheses concerning alternative incorporations of a given constituent into larger phrases.
- * To allow sharing of network representations among competing hypotheses.
- * To maintain intermediate results and hypotheses during the question answering process.
- * To define hierarchies of local contexts for discourse analysis.

Details concerning the encoding of various types of information in partitioned semantic networks are presented below.

B. THE ROLE OF SEMANTIC REPRESENTATION

Before considering the actual formalisms and conventions of the semantic representation, it will be helpful to gain a perspective on the utilization of the representation by various speech understanding system components. More comprehensive discussions concerning how each component uses partitioned semantic net structures are contained in other chapters of this report. Here, the goal is simply to provide a summary of the various interactions between the components and the network while processing an input.

Connections to the partitioned semantic network from other system components are pictured in Figure V-1 by broken lines. During system operations, the actual flow of semantic information is realized as the transfer of pointers into the net from one system component to another. The flow of network pointers, and of system control, is indicated in the figure by arrows. Heavy arrows labeled 'exec' indicate that control flows through the speech executive and not directly from one component to another.

As Figure V-1 shows, the network itself includes an encoding of knowledge about the domain of discourse. This encoding, called the domain model, includes descriptions of the objects and situations in the external world. All semantic processing performed by the system builds upon this model. For the ship domain, the model holds information about various ships and their properties.

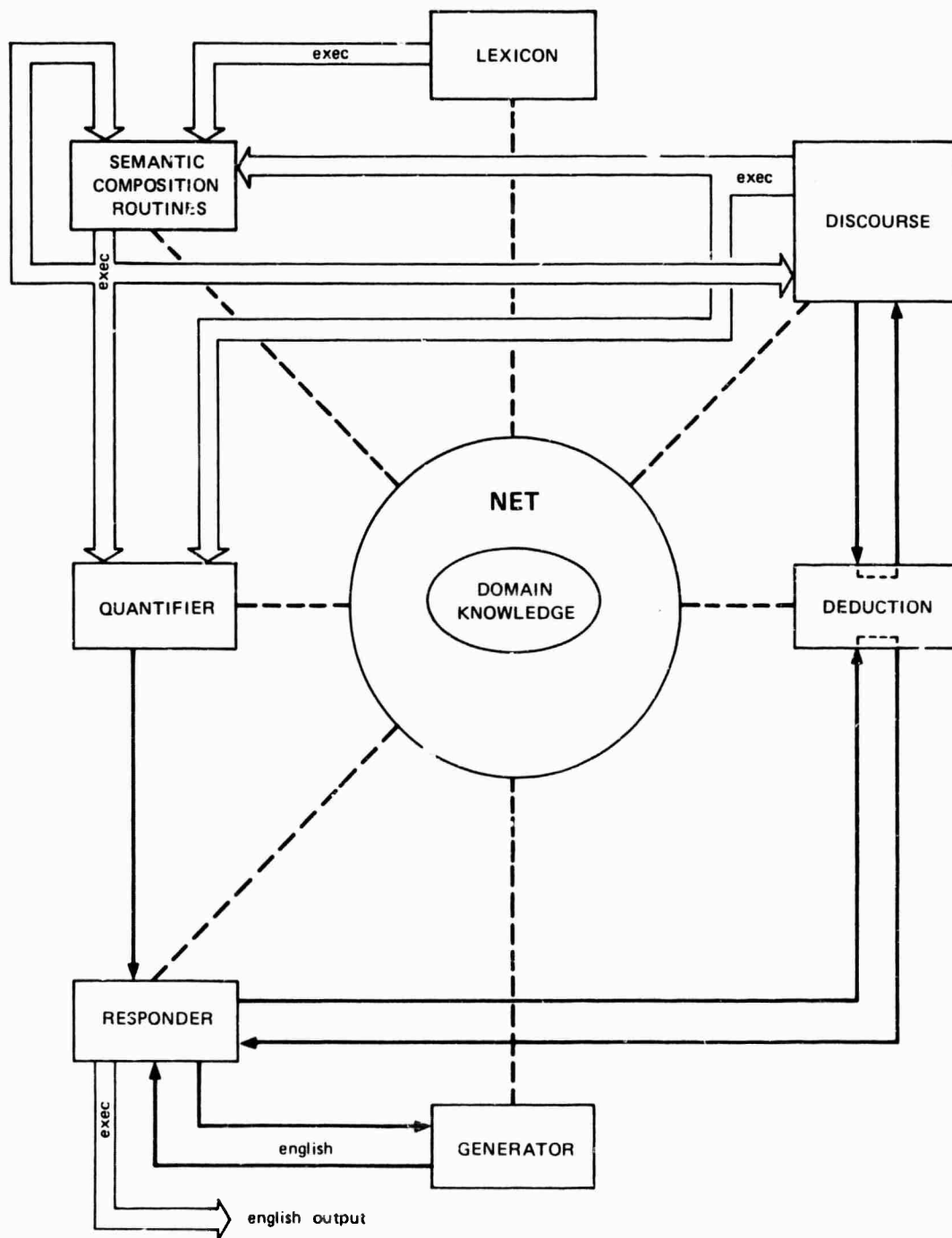


FIGURE V-1 FLOW OF SEMANTIC INFORMATION

The domain model serves three primary functions in the speech system. First, it is the source of information for answering user queries once the queries have been understood. Second, when language definition rules administered by the system executive discover a sequence of input utterance constituents that are syntactically capable of combining to form a larger phrase, knowledge from the network model is used to judge the feasibility of unifying the constituents to form a larger unit that has a meaningful interpretation in the task domain. In this capacity, the domain model serves as a parse time semantic filter. Third, the domain model serves as a foundation upon which structures encoding new inputs are built. This use of the model is in keeping with the fundamental principle that inputs are understood through an appeal to existing knowledge.

As anchor points for language understanding, the encoding in the network model of those concepts in the domain that can be referenced directly by individual words are referred to by records in the system's lexicon.* It is through these representations of basic concepts that interpretations of all inputs are understood. As the system executive attempts to combine lexical items into phrases and phrases into larger phrase units, references to the network are passed by the executive from the lexicon to the semantic composition routines. In the composition routines, the network structures referenced by phrase components are used in the construction of new network structures that represent the

* See the discussion of the lexicon in Chapter II.

meanings of the composite phrase. These new structures encode new instances or new combinations of concepts in the original network (domain model). Information concerning these newly created structures is then given back to the executive for use in combining the new phrase with other constituents (from the lexicon or composition routines) to form still larger structures. This process continues to combine network structures and to extend the net until an interpretation for the entire utterance has been constructed. The interpretation takes the form of a network fragment that is anchored to concepts in the original domain model but that is nevertheless external to and distinct from the domain model.

During the construction of an interpretation of an input utterance, any output from the semantic composition routines that constitutes the internal semantic description of a definite noun phrase is given to the discourse component for determination of the referent. The resolution of definite noun phrases may be viewed as the substitution of one network structure for another. The input network indicates the description (with respect to context) of some object in terms of other concepts. The output provided by the discourse component is a pointer directly referencing the network encoding of the object itself.

For elliptical utterances (ones that do not form complete sentences), the output from the semantic composition routines indicating the interpretation of the fragmentary input is given to the discourse component for expansion into an interpretation of a complete statement

or query. Here, a newly created complete interpretation is substituted for the network encoding of a partial interpretation.

The network structures that are built up by the semantic composition routines and the discourse component to represent the meanings of utterances do not directly indicate the scoping of (either implicit or explicit) quantifiers that appear in the input. Rather, because the determination of scopes is highly context sensitive, this task is performed by the quantifier module only after a (still unquantified) interpretation has been assigned to the total input. The quantification process makes no changes in the topology of nodes and arcs in the network fragment encoding the interpretation of an input. Instead, scoping is accomplished by adding new partitioning to the existing structure.

Once a network structure encoding a fully quantified interpretation of an input has been formulated, (a pointer to) it is passed to the responder module. The responder examines the interpretation and determines an appropriate course of action for producing a response that will satisfy the user. For the current system, inputs are expected to be questions or commands requesting information. For these types of inputs, the responder generates appropriate calls to the logical deduction component to retrieve (or derive) requested information from the network encoding of the domain model. In the deduction component, both the request for information and the collection of known facts are expressed in network notation.

The responder component can produce answers to YES/NO questions directly from the output of deduction. However, the deduction component answers WH questions by returning pointers to nodes in the semantic network that encode the appropriate answers. The responder component passes such pointers to the English generator, which produces phrases or sentences answering the original questions in ordinary language. The responder component then returns these English strings to the user as output.

To summarize, the original net encodes a model of the external world that provides the seeds for subsequent understanding. Translation involves building up network structures on top of this base net to represent the meanings of individual inputs. Answers to user questions are found by matching the net fragments encoding the translation of queries against the network encoding of domain knowledge in the system. When an answer is found, its network encoding is translated into English.

C. BASIC NETWORK NOTIONS

In its simplest form, a semantic network consists of a collection of nodes interconnected by an accompanying set of arcs. Each node represents an object (a physical object, situation, event, set, and others), and each arc represents an instance of a binary relation. Typical of the binary relations used in networks are set membership and 'deep case relations.' A deep case relationship is a relationship between a situation (or other gestalt concept) and a participant in the situation. For example, there is an 'obj' case relationship between an owning situation and the object that is owned. (The notion of a 'deep' case, which is a relationship between semantic objects, contrasts with the notion of a 'surface' case, which is a relationship between syntactic units.)

1. A PRELIMINARY EXAMPLE

Figure V-2 provides an indication of how the interconnections among nodes and arcs may be used in the encoding of knowledge. At the top of the figure is the node 'UNIVERSAL'. (Single quotes denote node names.) This node represents the set UNIVERSAL, the universal set of objects. Arcs labeled "s", called "s arcs", are used to indicate subset relationships that exist between UNIVERSAL and other sets. In particular, the s arc from 'CORPORATIONS' to 'UNIVERSAL' indicates that CORPORATIONS, the set of all corporations, is a subset of UNIVERSAL. (Again note the use of single quotes: 'CORPORATIONS' is a

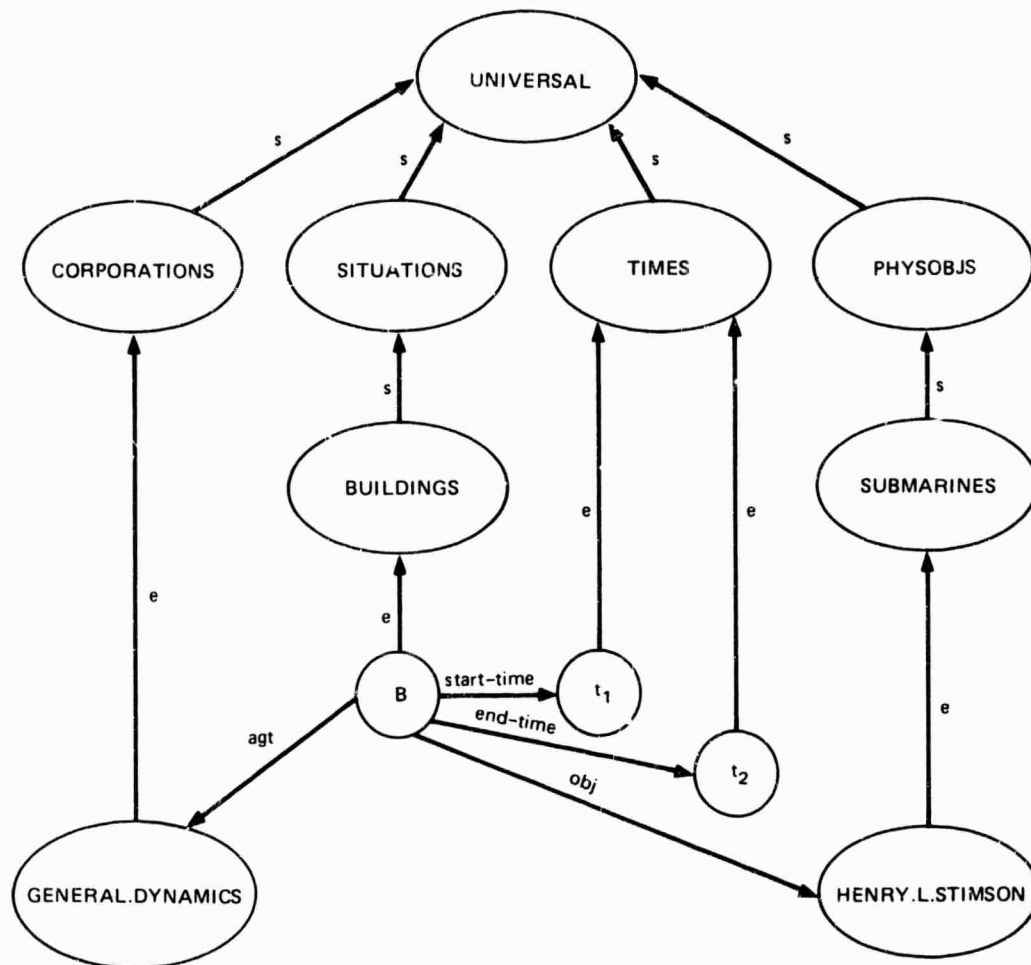


FIGURE V-2 AN EXAMPLE SEMANTIC NETWORK

node that represents CORPORATIONS.) Similarly, SITUATIONS, TIMES, and PHYSOBSJS (the set of all physical objects) are also indicated as being subsets of UNIVERSAL. At the next lower level, SUBMARINES, the set of all subs, is shown to be a subset of PHYSOBSJS.

Set membership is encoded in the network through the use of "e arcs". For example, the e arc from node 'Henry.L.Stimson' to node 'SUBMARINES' indicates that the Henry.L.Stimson is an element of SUBMARINES and is thus some particular sub. Similarly, General.Dynamics is a corporation and t1 and t2 are instants in time.

The node 'B' represents an element of the set BUILDINGS, the set of all building situations in which an agent constructs an object over some time period. (BUILDINGS is not the set of all roofed and walled structures.) In turn, BUILDINGS is a subset of SITUATIONS, the set of all static conditions and dynamic events. For the particular situation B, General.Dynamics is the agent that built the object, the Henry.L.Stimson, during the period from time t1 until t2. The components of situation B are associated with it through deep case relationships, which are encoded by case arcs emanating from node 'B'. For example, the agent of situation B is indicated by the agt arc from 'B' to 'General.Dynamics'. The other deep case relationships are encoded by obj, start-time, and end-time arcs.

2. RESTRICTIONS ON NODES AND ARCS

Proponents of network structures have adopted a number of different conventions concerning what types of concepts may be encoded by nodes and what types of relationships may or should be encoded by arcs.* In creating encoding structures for the SRI speech understanding

* For a useful perspective on these issues, see Woods, 1975.

system, an attempt is always made to use constructs that are easily understood by appealing to such familiar mathematical systems as set theory, predicate calculus, and case logic. (See Bruce, 1973, for a description of case logic.) The SRI system places no restrictions on the types of objects that may be represented by nodes. However, arcs are restricted to the encoding of hierarchical (element and subset) information and case relationships. Arcs are never, for example, allowed to encode relationships, such as ownership, that are time bounded.

The reason for this restriction on arcs arises from the fact that arcs are much less flexible than nodes. Arcs directly relate only three pieces of information (from-node, to-node, and arc label), and one of these (the label) is not back-indexed (i.e., most network schemes provide no easy mechanism for finding all arcs of a given label). Also, (omitting the new concept of a space, presented below) other network structures cannot point to an arc. Thus, for example, an arc does not have enough handles to relate the concept of ownership with an agt, obj, start-time, and end-time.

In contrast, a node may interrelate an arbitrary number of concepts simply by using multiple outgoing case arcs. Moreover, a node may be pointed to by an arc and hence may be modified by the arc and its from-node. When used to encode a relationship, a node will typically have an e arc to the set of all relationships of the same type (e.g., an e arc to 'OWNINGS') and case arcs pointing to each of its known

participants. All such relationships are members of the set SITUATIONS. Each member of this set relates some state of affairs or state of flux in affairs. Interpreted quite broadly, the elements of this important set each encompass the idea of a circumstance, or a set of circumstances, or an event, or a set of related events with their corresponding changes in circumstances, or a relationship, or a set of interdependent relationships.

Although some network systems (e.g., Sowa, 1976) attach significance to node labels, we do not. (Indeed, in the actual implementation, most nodes have no labels and are referenced only by location.) Names such as "SUBMARINES" may have significance to the system users but mean nothing to the network system itself.

Some network systems have a fixed number of arc labels with each having a special meaning to the network processor. While hierarchy encoding arcs are especially known to our processor, case names may be invented freely and do little more in the network than distinguish the various component parts of a complex object from one another. (The use of case arcs in language translation is a separate issue.) Routines that retrieve information from the network are actually made more efficient by the use of more case names, since an increase in the number of cases decreases the 'grain' size for searches.

3. THE HIERARCHICAL TAXONOMY

The presence of e and s arcs in a network serves to taxonomize the concepts represented by the various nodes in hierarchical form and is a key feature of the SRI notation. The significance of the taxonomy lies in the fact that many sets have associated with them a collection of properties common to all of their members. Any property that is characteristic of ALL members of a given set may be described at the set level and need not be repeated in the encoding of each individual set member. This set level encoding leads to great savings in storage. The actual encoding of properties common to all members of a set is dependent on the use of quantification, which is discussed later. Of importance here is the realization that the knowledge of whether or not an item belongs to a given set is of central relevance in question answering and fact retrieval.

To enhance the precision of the network encoding of taxonomies, the standard set-theory notions of set membership and set inclusion, which are expressed by e and s arcs, may be supplemented by the more restrictive concepts of disjoint subsets and distinct elements.

When certain subsets of a given parent set are disjoint (i.e., have no elements in common), this fact may be represented by ds arcs. As Figure V-3 shows, suppose that 'X', 'Y1', 'Y2', ..., 'Yn' are nodes in some particular network N representing the sets X, Y1, Y2, ..., Yn, respectively, and that for every i from 1 to n there is a ds arc

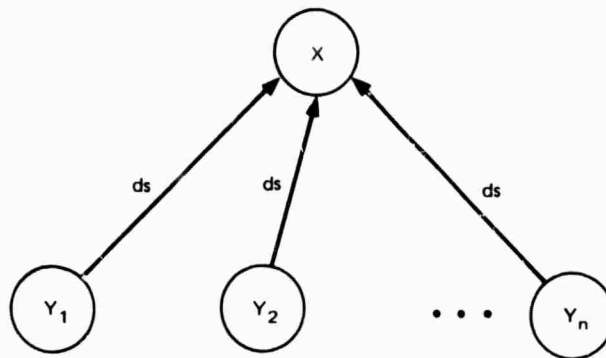


FIGURE V-3 ABSTRACTED USE OF ds ARCS

from ' Y_i ' to ' X '. The network N indicates that each Y_i is a subset of X . Further, the use of ds arcs rather than s arcs indicates that for unequal i and j , the intersection of Y_i and Y_j is the empty set. [Note: The empty set is the only set that is disjoint from itself. Thus, the set of unicorns (which in extension in the real world is the empty set) is disjoint from the set of chickens that have teeth (which also is empty).]

Whenever the need arises to show that the intersection of two sets S_1 and S_2 is nil, a parent set S_0 (which is the union of S_1 and S_2) may be constructed. S_1 and S_2 are then indicated to be disjoint subsets of this set.

If u and v are known to be elements of some parent set, it does not necessarily follow that u and v are distinct objects. That is, the symbols " u " and " v " may be references to the same object. To show that two nodes representing elements of a common set do in fact

represent two distinct entities, de arcs may be used. As shown in Figure V-4, suppose 'X', 'Y1', 'Y2', ..., 'Yn' are nodes in some network N representing the set X and the items Y1, Y2, ..., Yn, respectively. Suppose further that for every 'Yi' there is a de arc from 'Yi' to 'X'. The network N then indicates that each Yi is an element of X. Further, for unequal i and j, Yi is distinct from Yj.

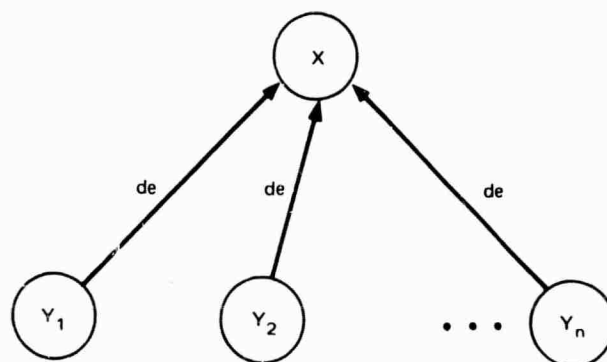


FIGURE V-4 ABSTRACTED USE OF de ARCS

To see the useful interplay between de arcs and e arcs, suppose Tom, Dick, and Harry went for a drive, and the driver wore a red cap. Tom, Dick, and Harry are distinct elements of the set of people who went for the drive, and their membership in the set would be recorded by three de arcs. The driver is also in this set, but could be any one of the three. Using a normal e arc to show the membership of the driver allows information about the driver (e.g., he wore a red cap) to be recorded while maintaining the uncertainty as to which of the three set members the driver really is.

The use of e, s, de, and ds arcs in a more extended example is shown in Figure V-5. This network indicates that CARRIERS, SUBMARINES, and P.SHIPS are all subsets of SHIPS. CARRIERS and SUBMARINES are indicated (by the ds arcs) to be disjoint sets. However, P.SHIPS is simply indicated to be a subset of SHIPS (by an s arc) and thus may intersect with CARRIERS or SUBMARINES or both.

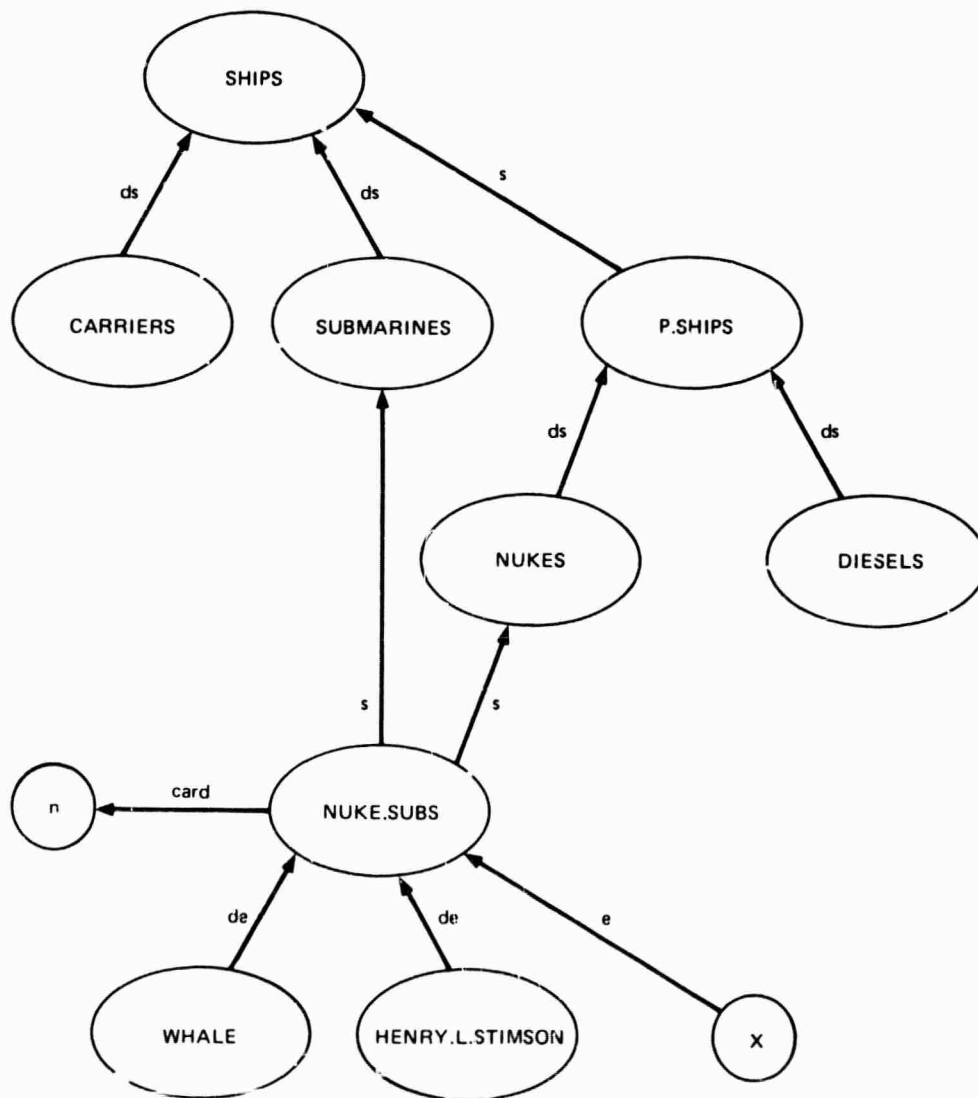


FIGURE V-5 THE USE OF ds AND de ARCS

P.SHIPS is itself shown to have the disjoint subsets NUKES (the set of nuclear powered ships) and DIESELS, whose intersections with CARRIERS and SUBMARINES might or might not be empty. (Figure V-5 actually indicates that the intersection of SUBMARINES and NUKES must include set NUKE.SUBS, which has at least two members.) The node P.SHIPS was used in this network solely to allow two independent divisions of SHIPS into disjoint subsets. To show the equivalence of SHIPS and P.SHIPS, an s arc may be drawn from 'SHIPS' to 'P.SHIPS'.

Set NUKE.SUBS is a subset of both SUBMARINES and NUKES. Its members include the Whale, the Henry.L.Stimson, and X. The Whale and the Henry.L.Stimson are known to be distinct. X might be either of these or yet some other nuclear sub.

The cardinality of the set NUKE.SUBS is indicated by the card arc* from 'NUKE.SUBS' to 'n'. If n were 2, then it would be possible to deduce that X is either the Whale or the Henry.L.Stimson. The very ambiguity of X (as shown in the Tom, Dick, and Harry example above) is attractive for some applications.

The use of ds and de arcs increases the power of the taxonomy by making it possible to prove negative assertions. For example, with CARRIERS and SUBMARINES known to be disjoint, it is possible to show

* Cardinality should probably be encoded by a node rather than an arc. This node, which would represent the relationship between a set and its cardinality, would have an e arc to 'HAS-CARDINALITY', a set arc to the set node, and a num (number) arc to the number node. The card arcs of the figures should be thought of as abbreviations for this larger structure.

that the Whale (or X) is not a carrier. Information about nonintersection and nonequivalence can be encoded by other means, but the de and ds arcs allow much of this information to be encoded for the price of the hierarchical information alone.

D. PARTITIONING

1. SPACES

To add a new dimension to the organizational and expressive power of semantic networks, the basic concept of a network as a collection of nodes and arcs may be extended to include the notion of partitioning (see Hendrix, 1975a,b). The central idea of partitioning is to allow groups of nodes and arcs to be bundled together into units. Each such bundle is defined by a new network construct called a "space". Spaces are fundamental entities in partitioned networks, on the same level as nodes and arcs.

Every node and every arc of the network belongs to (or lies in/on) one or more spaces. Spaces are fully cross-indexed with nodes and arcs. Given a space, all nodes and arcs lying within it are immediately determinable. Likewise, given a node or arc, all spaces on which it lies are directly available. Nodes and arcs of different spaces may be linked, but the linkage between such entities may be thought of as passing through boundaries that partition spaces. Nodes and arcs may be created in (initially empty) spaces, may be transferred

or 'copied' (at a fraction of creation cost) from one space to another, and may be removed from a space.

Spaces are useful in a variety of applications, including the encoding of quantifiers and other higher-order predicates. An important application of spaces in the SRI speech understanding system, which may be helpful to consider as an introduction to the partitioning concept, is in grouping together subparts of a semantic network that are capable of being expressed by a single syntactic unit. For example, Figure V-6 shows a network containing three spaces, two of which correspond to syntactic units. Each space is represented by a rectangle that contains the name of the space in the upper right corner. Thus, space S1 is at the top of the figure. Diagrammatically, a node or arc is indicated as belonging to a space if its label is written within the rectangle associated with the space. So, node 'C' and the e arc from 'C' to 'CORPORATIONS' lie only in S2. Spaces S1, S2, and S3 may be given concrete interpretations in the context of the sentence

"A corporation C built a submarine S."

Space S1 encodes background information (about corporations, building events, and submarines) for the understanding of this sentence. Space S2 encodes "a corporation C", the information that would be conveyed by the syntactic subject of the sentence. Space S3 encodes a building event in which a submarine S is the object of the building. This corresponds to the verb phrase of the sentence ("built a submarine S"). Figure V-6 does not in fact indicate that C was the agent in building event B, but this omission is corrected below.

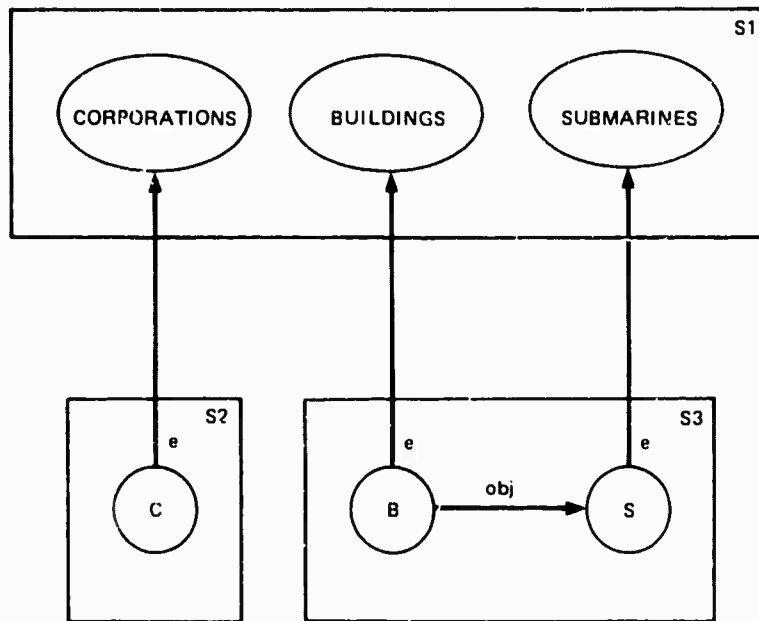


FIGURE V-6 SPACES SHOWING SYNTACTIC GROUPINGS

2. VISTAS

In using partitioned networks, it is often convenient to combine several spaces to form a composite bundle of nodes and arcs representing the aggregate of the bundles of the individual spaces. Such a combination of spaces is called a "vista". Most operations involving a partitioned semantic network are performed from the vantage of one of these vistas with the effect that the operations behave as if the entire network were composed solely of those nodes and arcs that lie in the spaces of the given vista. All structures lying outside the vista are ignored. To use an analogy with vision, when viewing the

network from a given vista V ; only those nodes and arcs are visible that lie in one of the spaces comprising V .

The mechanics of partitioning allow vistas to be created freely from arbitrary combinations of spaces. However, this freedom is seldom used. Rather, vistas are typically created in a hierarchical fashion by adding one new space to an existing vista or by adding a new space to the union of multiple existing vistas. The new vista created in this fashion inherits a view of the information in the parent vista(s) and adds a new space for extending locally available information without altering the view of the parent(s). Such hierarchically created vistas are analogous to programming contexts with global and local variables. Information structures in the spaces of the parent vista(s) are global relative to the new space, while structures created in the new space are local.

If space S_i is created to be the local space of vista V_i following the hierarchical pattern of vista growth described above, then S_i is called the "bottom space" or "lowest space" of V_i , and V_i is called the "orthodox vista" (or, when there is no confusion, simply the "vista") of S_i . Since S_i and V_i are so closely related, it will be convenient to talk about "viewing the net from the vantage of S_i " when the viewing is actually from V_i .

When new vistas are created hierarchically, they form a partial ordering of viewing capability. An example of such a partial

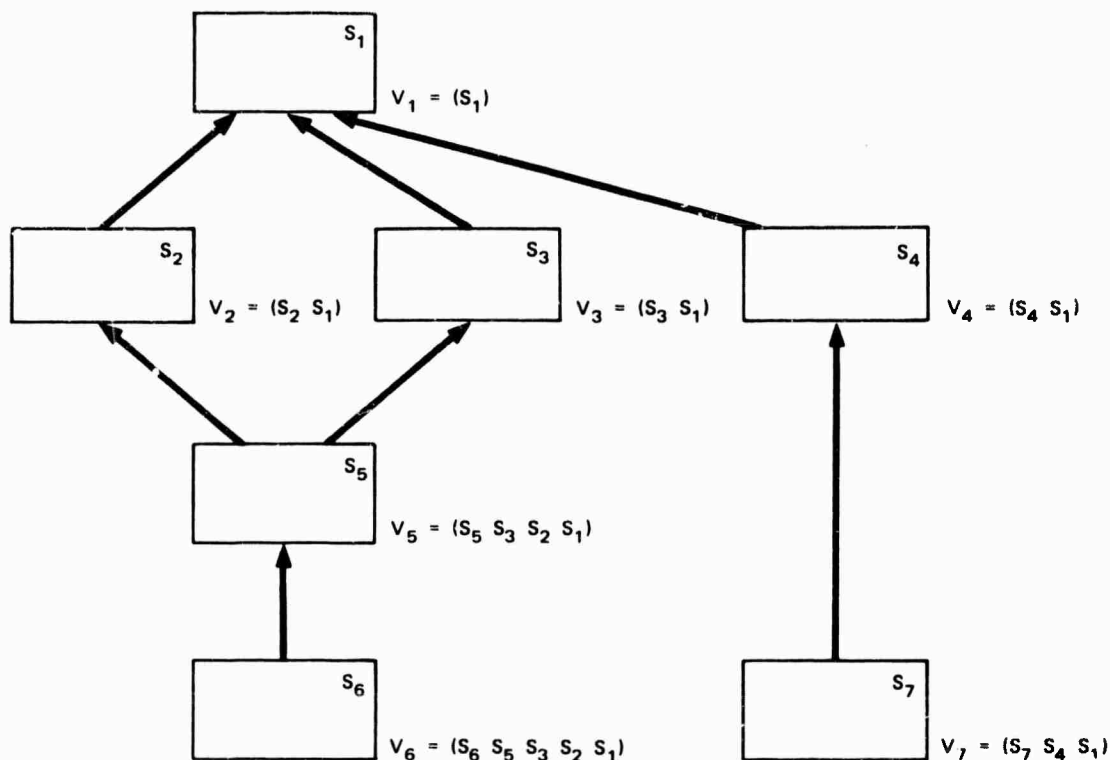


FIGURE V-7 ABSTRACTION OF VISTA ORDERING

ordering is depicted in Figure V-7. The spaces that are included in the various vistas are represented by rectangles as before. To the right of each rectangle is a list notation (vistas are actually implemented as LISP lists) indicating the orthodox vista of the space. Heavy arrows indicate the inheritance of viewing capability. That is, from any point in the partial ordering, information is visible on any space that may be reached by following up heavy arrows.

Space S1 at the top of the figure is associated with orthodox vista V1, which contains only space S1. From the vantage of V1, only the information in S1 is visible. Since this information is the most global information in the hierarchy, S1 and V1 are called the "root space" and "root vista", respectively.

The orthodox vista of S2 is V2, which contains both S2 and S1. Thus, from the vantage of V2, all the information in both S2 and S1 is visible. However, the information in S3 is not visible from V2 (except to the extent that S1 or S2 contain some of the same nodes and arcs as S3). From the vantage of V5 it is possible to see all the information in both S2 and S3, as well as the information in S5 and S1.

Figure V-3 provides some indication of how vista hierarchies may be used in a practical way. Again, the heavy arrows indicate which spaces are included in the (orthodox) vista of any of the spaces. From the vista of space VP, it is possible to see information on spaces VP, V, NP2, and BACKGROUND. Thus, from the vantage of VP, it is possible to see the background information and the structures used in creating a network interpretation of the verb phrase (VP) in the sentence "A corporation built a submarine." This view includes the information of space V (which encodes the verb alone), space NP2 (which encodes the direct object alone), and space VP (which encodes the relationship between the verb and object). From this same vantage, the structures in spaces NP1 and S are invisible.

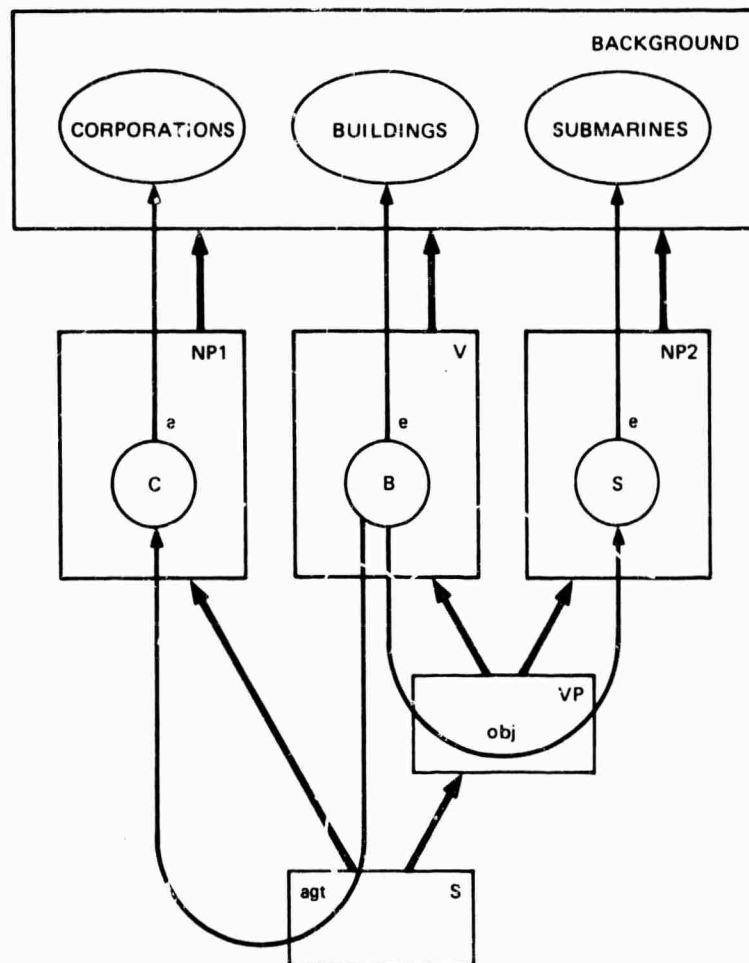
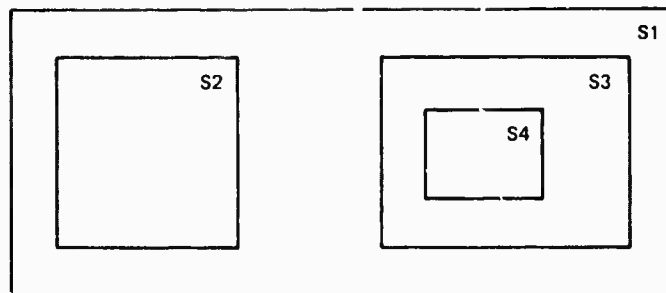
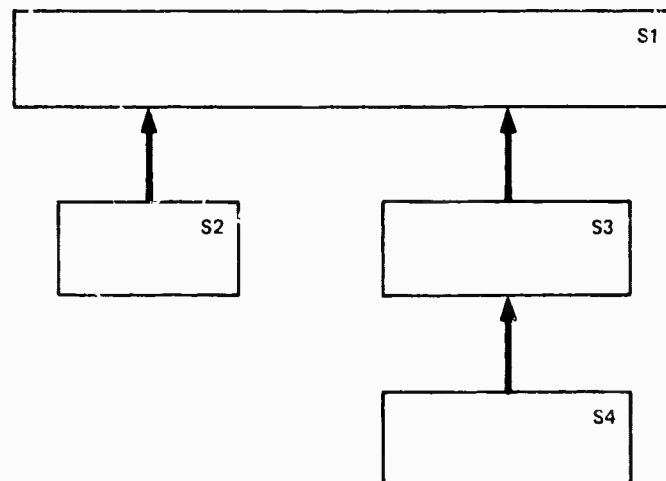


FIGURE V-8 USE OF VISTAS IN SYNTAX ENCODING

In subsequent diagrams, when a rectangle representing a space S is drawn completely within a rectangle representing a second space S' , then the orthodox vista of S is an extension of the orthodox vista of S' . For example, A and B in Figure V-9 represent equivalent structures. If two rectangles overlap, but neither contains the other,



(a)



(b)

FIGURE V-9 EQUIVALENCE OF ENCLOSURE AND HEAVY ARROW NOTATION

then structures appearing in the overlap lie on both spaces. Examples of such overlaps occur in the section on quantification below.

3. SUPERNODES

By bundling together a collection of structures, a space may be used to represent a complex concept that is in some way related to the aggregate of information encoded by its internal nodes and arcs. For example, a certain space S might bundle together a collection of nodes and arcs which, when taken together, represent the set of things that some person has reported to be true, or believes to be true, or wishes to have happen. Each node and each arc represents some aspect of the belief (report, wish), but only the space represents the belief structure (report, wished-for condition) itself.

Since it is often necessary to relate other concepts in the semantic network to the complex concept encoded by a space, spaces are (but only when necessary) given all the properties normally associated with nodes. In particular, arcs from ordinary nodes may point to spaces. This situation is shown abstractly in Figure V-10. Node 'X' represents a believing situation in which the believer (agt = agent) is JOHN and the thing believed (thm = theme) is a complex of information encoded by space S. The structures inside S (omitted in the figure) may be thought of as describing a hypothetical world (HYPO.WORLD) in which JOHN believes. (How to represent what the system itself believes it believes is an interesting, solvable problem that the reader may wish to consider after completing this section.)

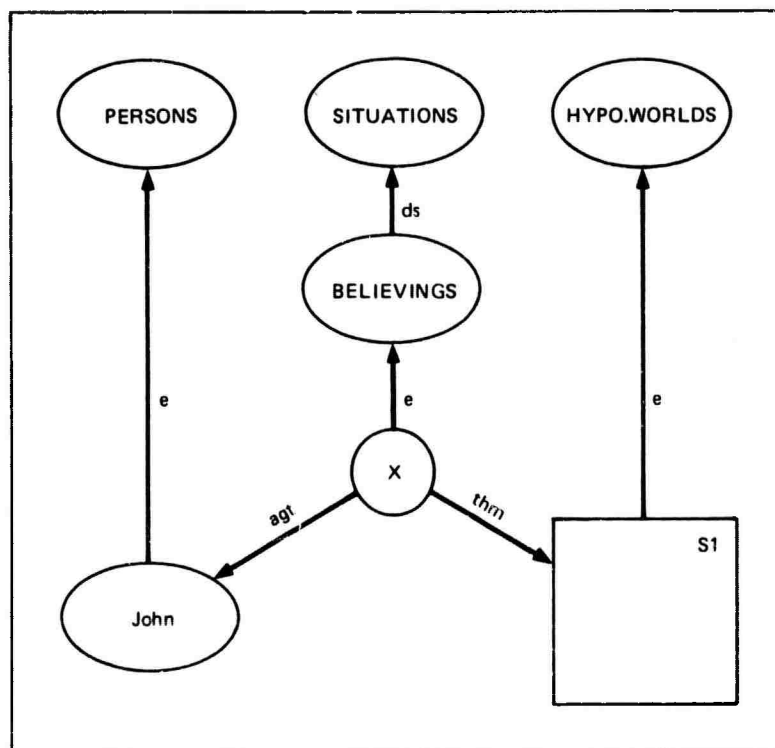


FIGURE V-10 THE BELIEFS OF JOHN

When spaces are given node-like properties, they are called "supernodes". In the SRI speech understanding system, the primary use of supernodes is in encoding higher-order structures (including logical connectives and quantification), which is the subject of the next section.

E. HIGHER-ORDER STRUCTURES

A primary reason for developing the concept of network partitioning was to provide an efficient and uniform mechanism for dealing with higher-order logical constructs. This section discusses the application of partitioning to the encoding of logical connectives, quantifiers, questions, and other concepts that are encoded only clumsily (if at all) by networks lacking partitioning.

In discussing higher-order encoding structures, it is important to bear in mind that logical connectives, quantifiers, and other mathematical formalisms are simply tools that are useful in the construction of models. The goal of developing a network encoding structure is not to represent higher-order mathematical formalisms (although this is possible and is important in some task domains) but rather to represent the types of knowledge that typically require such formalisms for their representation in other formal logics.

It is also important to bear in mind that formal statements may be regarded from either of two perspectives. From the first perspective, a statement represents a proposition whose truth or falsity may be tested in arbitrary worlds W . From the other perspective, a statement designates a set of conditions. This designation in turn provides a partial description of all worlds in which those conditions hold. In building network models, the latter is usually the perspective from which network formalisms are most easily understood.

1. LOGICAL CONNECTIVES

Rather than cover possible network encodings for a wide variety of logical connectives (there are 16 binary connectives alone), attention will be focused on CONJUNCTION, DISJUNCTION, and NEGATION. From the network encodings of these three, encodings for any logical connective may be constructed. (Indeed, NEGATION in combination with either CONJUNCTION or DISJUNCTION forms a logically complete set.) The IMPLICATION connective, because of its importance in the network encoding of quantification, also will be considered.

a. CONJUNCTION

As the first logical connective, consider CONJUNCTION. From one perspective, a conjunction is a proposition that relates a number of component statements called 'terms' or 'conjuncts'. The proposition is itself true if and only if each of the individual conjuncts is true. From the standpoint of model building, it is useful to think of each conjunct as a description of some condition. The conjunction itself then becomes a complex description of the situation in which the conditions described by each of the individual conjuncts exist in unison. From either perspective, a conjunction bundles together multiple conjunctive terms. If the bundle is accepted (as being true or as providing a partial description of some world), then the various terms must be accepted collectively. Conversely, if any term in the bundle is not accepted, then the bundle itself is rejected.

The inherent bundling capability of spaces makes the space formalism a convenient mechanism for the network encoding of conjunction. In particular, a conjunction C may be represented by a space S upon which each conjunct of C (and only the conjuncts of C) is encoded as a net structure. Space S2 of Figure V-11, for example, encodes the conjunction "The Henry.L.Stimson was built by General.Dynamics AND the Henry.L.Stimson is owned by the U.S."

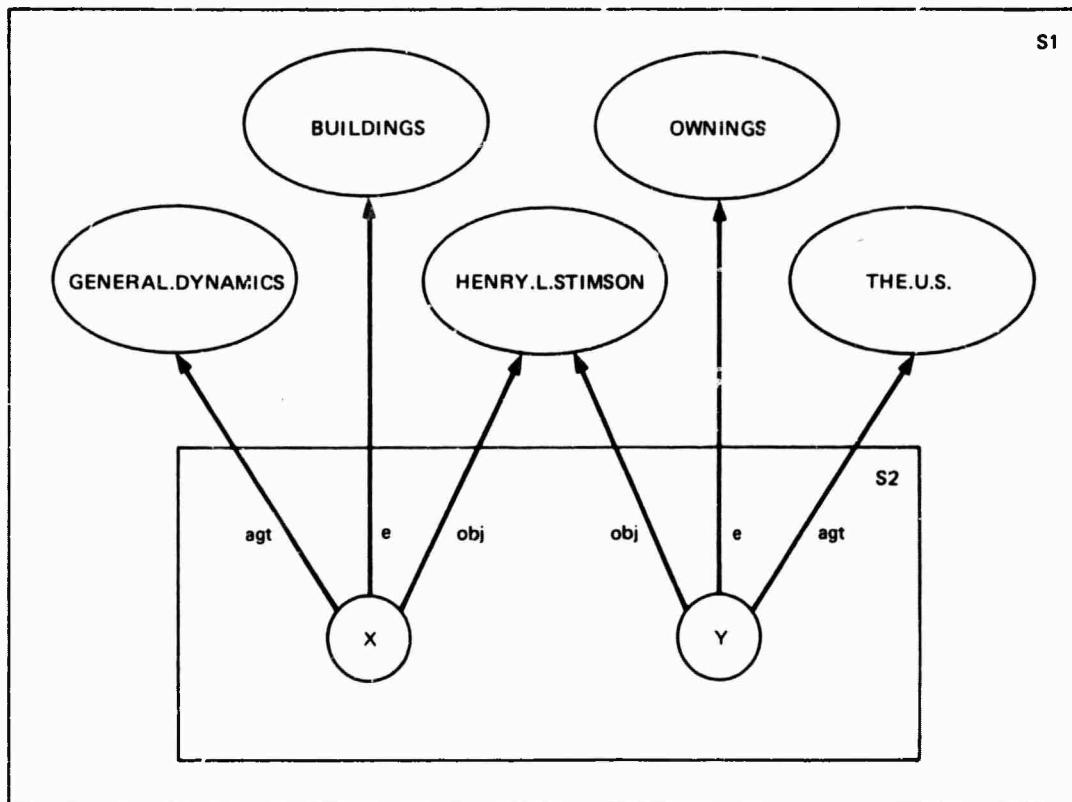


FIGURE V-11 THE CONJUNCTION "THE HENRY.L.STIMSON WAS BUILT BY GENERAL.DYNAMICS AND THE HENRY.L.STIMSON IS OWNED BY THE U.S."

The subordination of S2 under S1 in the viewing hierarchy is rather artificial and was done here solely for exposition. Except for delimiting the conjunction (X & Y), the structures of S2 might just as well have been encoded directly in S1. This ability to remove S2 after moving its structures to S1 is the network analog of the ability to remove the embedded parentheses in the formula

(A & B & (X & Y) & C)

to form

(A & B & X & Y & C) .

The merit of placing the terms of a conjunction on their own space will become apparent when the conjunction is dominated by a structure other than another conjunction (such as a disjunction).

b. DISJUNCTION

As the second logical connective, consider DISJUNCTION. Unlike a conjunction, which groups together a set of statements for consideration as a unified whole, a disjunction separates out a number of alternative statements. The disjunction is accepted into a belief system or model if any of the individual statements (disjuncts) is accepted. The inherent separating ability of spaces makes the space formalism a convenient mechanism for the network encoding of disjunctions. In particular, each of the n disjuncts of a disjunction D may be encoded on different spaces and so kept in (relative) isolation.

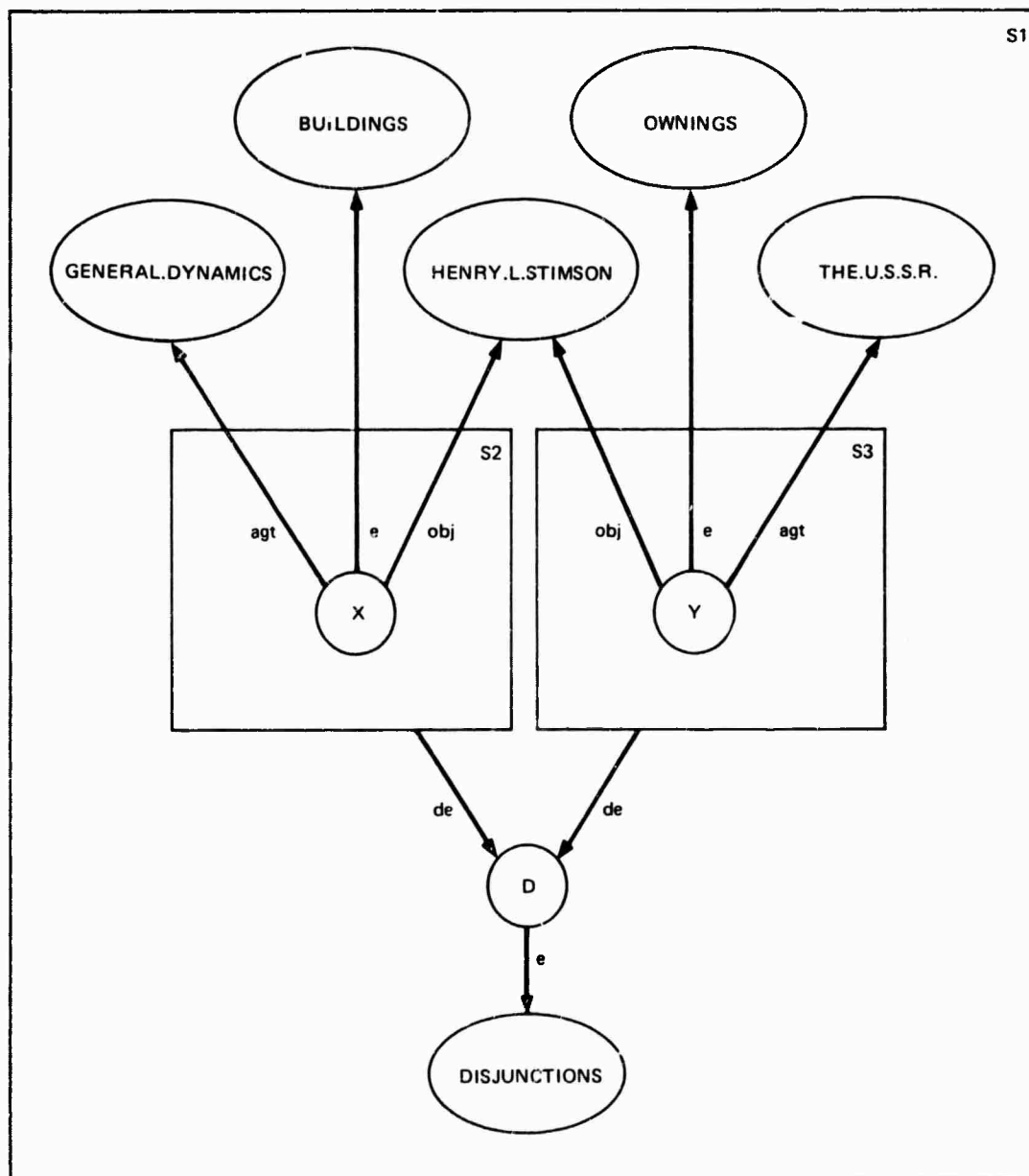


FIGURE V-12 THE DISJUNCT "EITHER THE HENRY.L.STIMSON WAS BUILT BY GENERAL.DYNAMICS OR THE HENRY.L.STIMSON IS OWNED BY THE U.S.S.R."

Figure V-12, for example, shows the network encoding of the disjunction $D = \text{"Either the Henry.L.Stimson was built by General.Dynamics, OR the Henry.L.Stimson is owned by the U.S.S.R."}$ Node 'D' represents the disjunction itself, an element of DISJUNCTIONS, the set of all disjunctions. The disjuncts of D are represented by spaces (supernodes) S2 and S3. Since a disjunction may be regarded as a SET of alternative disjuncts, the disjuncts of D are shown as distinct elements of D. Whenever a disjunction appears in the network, it is assumed that all members of the disjunctive set are explicitly encoded. [A more elaborate encoding scheme might encode a disjunction by a node 'D' with a case arc (a disjuncts arc) to a node 'S' representing the set of disjuncts. Such a structure would separate the notion of a disjunction from the notion of a set of statements. For the SRI speech understanding system, this distinction was not considered worth the extra structure.] Since the disjuncts of D are represented as spaces, each disjunct is an implicit conjunction (which might, however, "conjoin" only a single conjunctive term).

The entire disjunction structure is embedded in the conjunction encoded by S1. From the modeling perspective, S1 represents some world and each structure in S1 represents some object or situation that occurs in that world. So, in viewing the network from the vantage of (the orthodox vista of) S1, such entities as General.Dynamics and D are seen to occur. However, the structures in spaces S2 and S3 are not seen from the vantage of S1 and are thus not asserted in the world

modeled by S1. Since D does appear in the world of S1, it is known that the world of S1 includes the situations described by at least one of the disjuncts of D. If S2, for example, were included, then the modeled world would include all situations described by structures that are visible from the vantage of S2. This view includes structures in S2 and S1, but excludes structures in S3.

The encoding of "The Henry.L.Stimson is owned by either the U.S. or the U.S.S.R." that is shown in Figure V-13 has the following interesting feature. From the vantage of S1, it is possible to see that there is an owning situation involving the Henry.L.Stimson as an object. However, the agt arcs from 'X' are not visible in S1, so the owner is not known. But in viewing the network from the vantage of either of the disjuncts of D, the agent of the owning is specified. [There are several other ways to encode the information of Figure V-13, but the structure shown is one of the least expensive. Other methods include (1) using two nodes with e arcs to 'OWNINGS', and (2) making the agt some dummy node 'Y' with an e arc to a node 'S' that represents a set of cardinality two with distinct elements The.U.S.S.R. and The.U.S.]

c. NEGATION

The network encoding of NEGATION, like the network encoding of disjunction, uses the separating aspect of spaces. But rather than separating multiple alternatives, negation uses spaces to

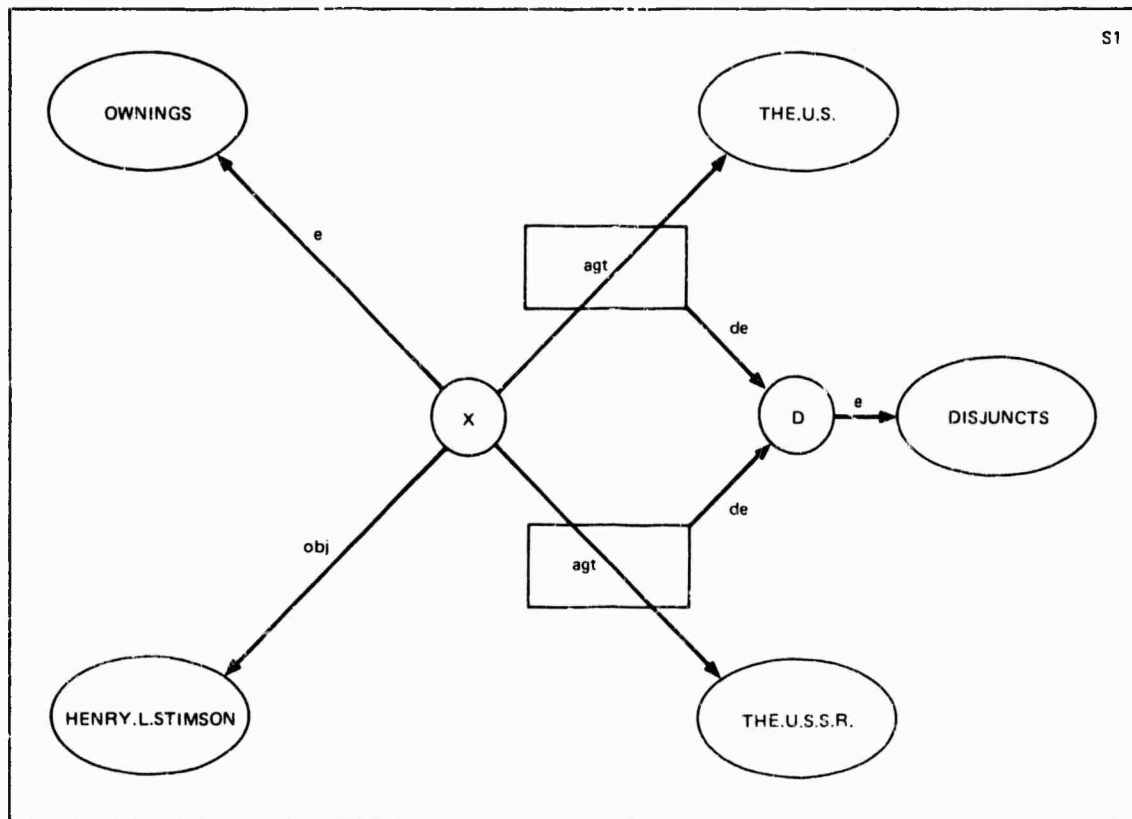


FIGURE V-13 THE HENRY.L.STIMSON IS OWNED BY EITHER THE U.S. OR THE U.S.S.R

separate the negative from the positive. Figure V-14 shows the network encoding of the negation "The U.S.S.R. does NOT own the Henry.L.Stimson." The negation, an element of NEGATIONS, is encoded by space (supernode) S2. S2 is an (implicit) conjunction describing a set of situations that cannot occur simultaneously in the context of the situations described in S1. As in the disjunction example, the negated structures inside S2 are not visible when viewing the network from the vantage of S1, although the negation itself is visible.

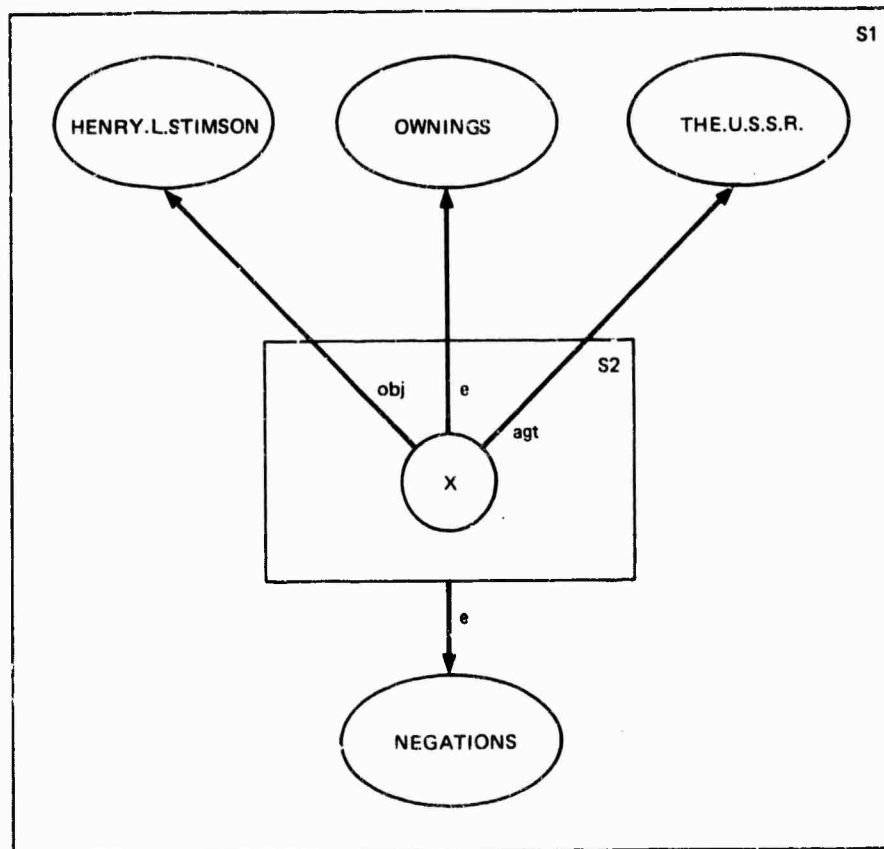


FIGURE V-14 THE U.S.S.R. DOES NOT OWN THE HENRY.L.STIMSON

d. IMPLICATION

Using the formalisms developed for conjunction, disjunction, and negation, it is possible to construct network encodings carrying the force of any logical connective. Note in particular that the implication

$$P \Rightarrow Q$$

is equivalent to

$\sim P \vee Q$

which uses only disjunction and negation. The network of Figure V-15 takes advantage of this transformation to express the implication

"If General.Dynamics built the Henry.L.Stimson,
then the U.S. owns it"

as the disjunction

"Either General.Dynamics didn't build the Henry.L.Stimson,
OR the U.S owns it."

Since the notion of implication is closely tied to the expression of general rules and, as will be seen subsequently, to the use of universal quantifiers, it is important that implications be expressed both simply and economically. As might be expected, if both P and Q are positive, it is more economical to express $\{P \Rightarrow Q\}$ directly as an implication than it is to transform it into a disjunction. (If $P = \sim R$, then $\{R \vee Q\}$ is the economical encoding. If $Q = \sim S$, then $\sim\{P \& S\}$ is economical. If $P = \sim R$ and $Q = \sim S$, then $\{S \Rightarrow R\}$ is economical.) An implicational encoding of the information of Figure V-15 is presented in Figure V-16. In the new figure, node 'I' encodes the implication. Each implication has two component parts, an ante (antecedent) and a conse (consequent), which are encoded as spaces. It is the situations of the ante that imply the situations of the conse. The positive alternative (S3) of disjunction D corresponds to the conse (T3) of I. The ante (T2) of I corresponds to the negated alternative (S4) of D.

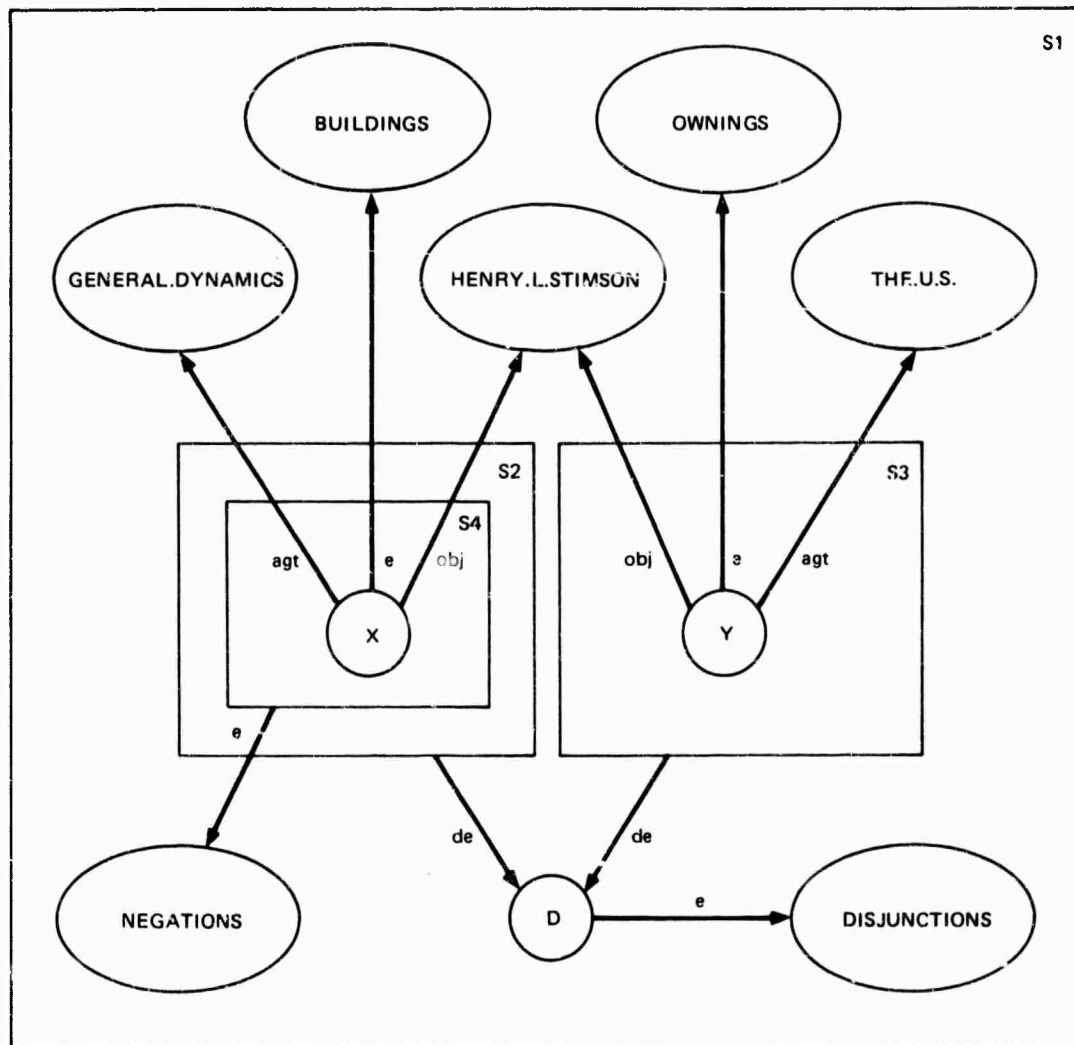


FIGURE V-15 EITHER GENERAL.DYNAMICS DID NOT BUILD THE HENRY.L.STIMSON,
OR THE U.S. OWNS IT

To further decrease the costs associated with the encoding of implications, the abbreviation shown in Figure V-17 may be used. Antecedents of implications are placed in the set IMPLICATIONS* and associated with their corresponding consequences

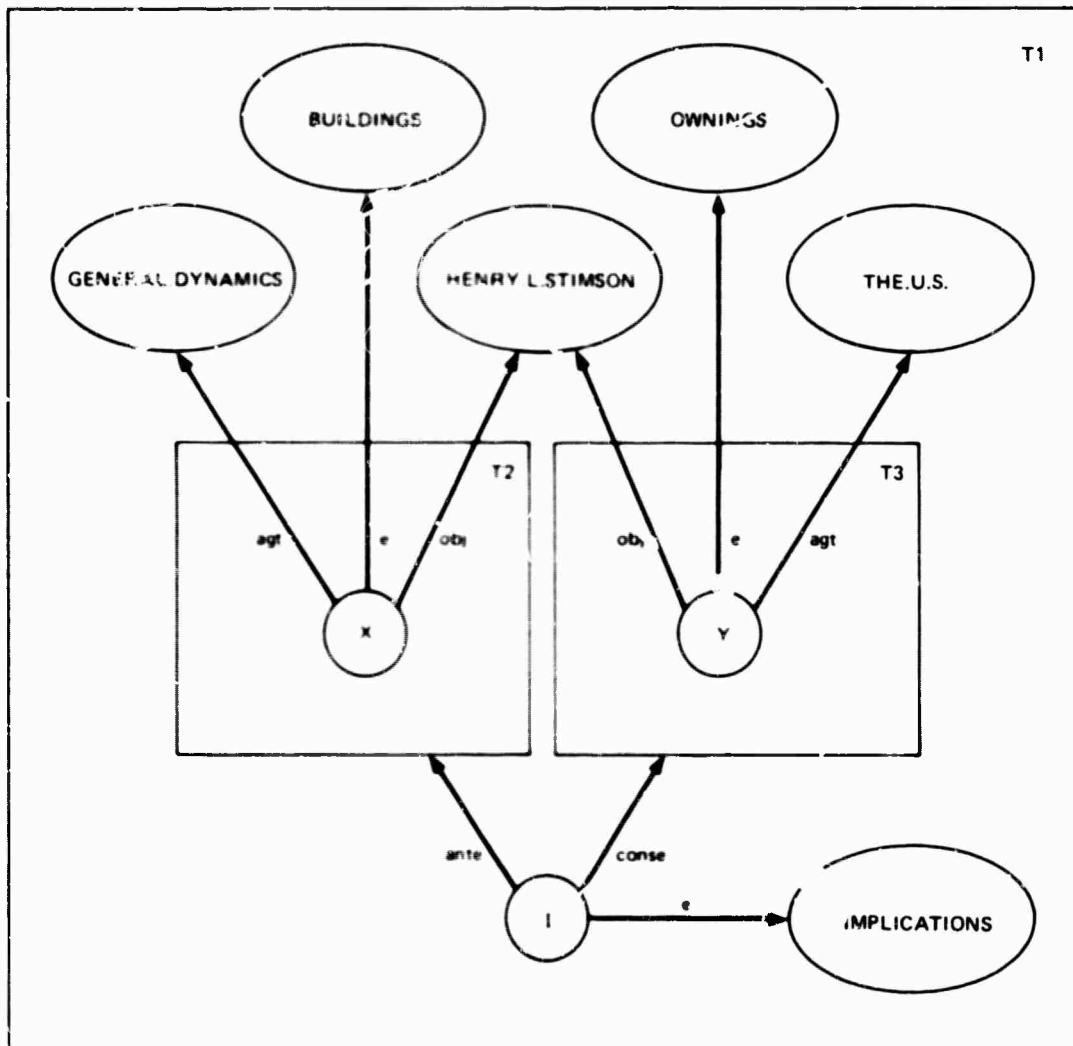


FIGURE V-18 IF GENERAL DYNAMICS BUILT THE HENRY L. STIMSON,
THEN THE U.S. OWNS IT

through conse arcs. Since the abbreviation is rather ugly and saves only one node and one arc per implication, it is probably worthwhile only in systems with severe storage limitations.

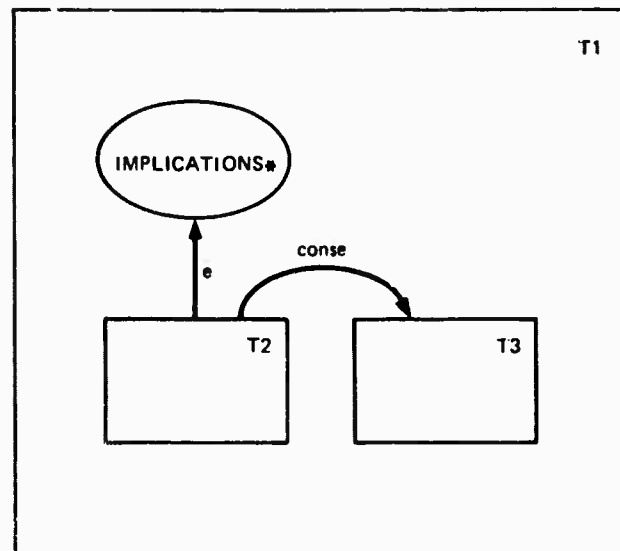


FIGURE V-17 COMPACT IMPLICATION NOTATION

e. SPACES AS CONJUNCTIONS

For understanding subsequent sections, it is important to look back at the spaces used in forming logical connectives and to realize that these spaces always act as conjunctions. CONJUNCTIONS are themselves simply encoded as spaces. DISJUNCTIONS are encoded by a set of spaces, each of which represents an alternative conjunction. NEGATIONS are encoded by spaces that represent negated conjunctions, and both the ante and conse of IMPLICATIONS are conjunctions.

2. QUANTIFICATION

In addition to handling the encoding of individual pieces of information and joining individuals by logical connectives, a system for representing knowledge should be able to deal with quantified information. Partitioning offers a number of alternatives for the encoding of quantified information. The more interesting of these use spaces to group together collections of universal or existential variables whose scopes are commutative. Nestings of scopes (and the dependency of existentials on higher universals) is then encoded by using an appropriate hierarchy of orthodox vistas. Details for two schemes using this general approach are presented in subsections below.

a. THE IMPLICIT EXISTENTIALS OF PROPOSITIONS

Before actually getting into the details of quantification in nets, it is necessary to consider one aspect of quantification that is often overlooked in dealing with formulas in first order predicate calculus. This aspect is the implicit existential quantification carried by propositions.

Letting the symbol "A" represent "FOR-ALL" and the symbol "E" represent "THERE-EXISTS," an ordinary formula in first order predicate calculus such as

$$\text{AxEy}[p(x,y)]$$

may be read as "for all x there exists a y such that $p(x,y)$." Although

it may appear that all the quantification information of this formula is encoded in the prefix (i.e., in "AxEy"), the proposition $p(x,y)$ itself implicitly encodes existential information. That is, proposition $p(x,y)$ proposes that THERE EXISTS a particular type of situation involving x and y .

One way of thinking about predicate p is that it models a set of situations S .^{*} Each situation in the set has two participants. To distinguish the participants, they may be given names such as "case1" and "case2." For any two entities x and y , $p(x,y)$ will be true if and only if there is a particular situation i in S such that x is the case1 of i and y is the case2. Thus, one interpretation of $p(x,y)$ is "there exists situation i , an element of S , whose case1 is x and whose case2 is y ."

Using the interpretation cited above, $p(x,y)$ carries at least four pieces of information:

1. THERE EXISTS i
2. i is an element of situation set S
3. the case1 of i is x
4. the case2 of i is y

^{*} There is a chicken-and-egg problem concerning whether p models S or vice versa. Although it is possible to think of S as being a set defined by predicate p , it is probably more useful and realistic to think of S as existing prior to p and to think of p as being a mathematical model of situation set S . To see this interpretation, consider the set of owning situations existing in our everyday world. This set exists whether or not anyone cares to invent a predicate called "owns" to model it. Note carefully that S is a set of situations and is not a set of n -tuples composing a formal mathematical relation. In particular, S is distinct from the set R of pairs (x, y) where (x, y) is in R if and only if $p(x,y)$. Predicates, sets of tuples and network structures only serve to model S .

For certain applications (as shown below), it is important to separate out one of these four pieces. Since predicate calculus notation bundles these pieces together, some new notational conventions are needed. Hence, let the notation

$$\langle q \ x_1 \ x_2 \ \dots \ x_n \rangle$$

indicate a situation of type q over the participants x_1 through x_n . For each situation type q , case names will be defined for each of the positions x_1 through x_n . Further, the q designating the situation type may be either the associated predicate (" p " in the example above) or the associated situation set (" S "). In discussions of predicates in which the associated set is not explicitly named, the capitalization of the predicate symbol may be used as the name of the set. Thus, were S not explicitly named, the set associated with p would be P .

Since a situation designator q may be either a predicate or set name, the expressions " $\langle p \ x \ y \rangle$ " and " $\langle S \ x \ y \rangle$ " are equivalent, both denoting a situation from the set S with case1 x and case2 y . Using situation notation, the proposition

$$p(x,y)$$

may be expressed as

$$E\langle S \ x \ y \rangle$$

(or, equivalently, as $E\langle p \ x \ y \rangle$). That is, $p(x,y)$ means "there exists $\langle S \ x \ y \rangle$," or "there exists an element of S with participants x and y ."

Thus, the formula

$$AxEy[p(x,y)]$$

may be restated in the form

$AxEyE\langle S \ x \ y \rangle$

which explicitly indicates the existential quantification of the situation.

Recalling that $E\langle S \ x \ y \rangle$ means that there exists an element i of S with case1 x and case2 y , the formula

$E\langle S \ x \ y \rangle$

may be thought of as a shorthand for

$EiE\langle e \ i \ S \rangle E\langle \text{case1 } i \ x \rangle E\langle \text{case2 } i \ y \rangle$

where $\langle e \ i \ S \rangle$ is the situation of i being an element of S and $\langle \text{case1 } i \ x \rangle$ is the situation of x being the case1 of i .

In existentially quantifying a situation $\langle q \ x_1 \ x_2 \ \dots \ x_n \rangle$, it has been assumed that the situation participants x_1 through x_n were either constants or already quantified. Should the expression $E\langle q \ x_1 \ x_2 \ \dots \ x_n \rangle$ ever appear with some x_i unquantified, it is to be interpreted as a shorthand for $Ex_iE\langle q \ x_1 \ x_2 \ \dots \ x_n \rangle$.

b. THE ORTHOGONAL PARTITIONING APPROACH TO QUANTIFICATION

(Note: This section presents an approach to quantification that was not used in the SRI speech understanding system, and may be skipped. However, the approach provides a clear encoding for branching quantifiers and a clean separation between quantification and the use of logical connectives.)

The most straightforward technique for encoding quantification based on partitioning is called the "orthogonal partitioning approach." By using this technique, the network is partitioned (at least) twice. One partitioning is used to encode logical connectives and similar structures. The other partitioning is used solely for the purpose of encoding quantification. A typical node in the network will lie on two spaces, with one space, called the "matrix" space, showing the node's relationship to the logical connectives and the other space, called the "quantification" space, showing the node's quantification. Similarly, arcs lie on two spaces.

The matrix spaces are arranged in the hierarchy that is used by the logical connectives (as discussed previously). The quantification spaces are arranged in a hierarchy such as that shown in Figure V-18. Each space is associated with either existential information (spaces whose labels begin with "E") or universal information (labels with "A"). At the top of Figure V-18 is space E0, encoding the top-level existentials. These are the system's constants. Directly below E0 are spaces associated with universals. Then at the next level, there are more existential spaces, and so on.

All of the semantic networks presented in illustrations thus far were intended to encode purely existential information, although no explicit indication of quantification was used. In the orthogonal approach to quantification, an explicit indication of the existential nature of the information in the previous nets could be

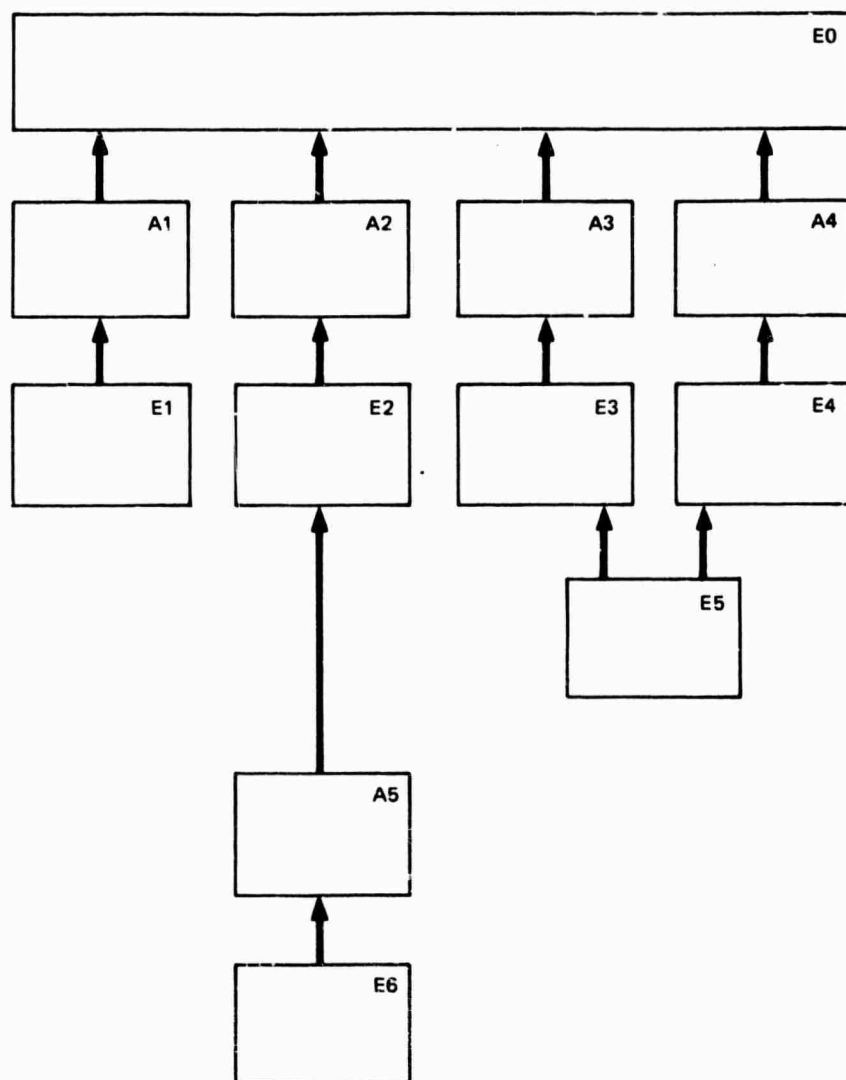


FIGURE V-18 A HIERARCHY OF QUANTIFICATION SPACES

realized by placing every node and every arc of these networks on space E0 of the quantification partitioning. Universal quantification is introduced by using vistas that contain more than the single space E0.

The vista (E1 A1 E0) is typical of the vistas in the quantification hierarchy. The nodes and arcs lying on E1 are to be

considered as existential variables that are dependent upon the universals specified at higher levels in the hierarchy. In particular, the structures of E1 are dependent on the universal structures of A1. So, if A1 contains x and E1 contains y, the encoded quantification might be expressed as

$$\text{AxEy}$$

Matrix spaces overlapping spaces of the vista of E1 will be presented shortly to complete the encoding of a quantified statement.

For a more complex quantification vista, consider the vista of E6. The existential structures on E6 are dependent upon the universal structures in A5 and A2. The existential structures of E2 are dependent only upon the universal structures in A2. If A2 contains structure u, E2 structure v, A5 structure w, and E6 structure x, then the quantification might be written as

$$\text{AuEvAwEx} .$$

The vista associated with E5 encodes a "branching quantification." For every A3 and A4, there exist E3, E4, and E5. But the E3 depend only on A3, and the E4 depend only on A4. The E5 depend on both A3 and A4. Hintikka (1974) presents an interesting discussion of branching quantifiers. One of his examples that fits vista E5 is "every writer likes a book of his almost as much as every critic dislikes some book he has reviewed." The disliked book depends only on the critic, the liked book only on the author, and the relation between the like and dislike on both the author and the critic.

To see how orthogonal partitioning is used in the encoding of an actual expression, consider

$$\text{AxEy}[p(x,y)] ,$$

which is encoded by the network of Figure V-19. Since two partitionings have been used, the figure presents two displays of the net with the top display showing the matrix partitioning and the bottom display showing the quantification partitioning. The quantification vista (E1 A1 E0) also appeared in Figure V-18.

Thinking of $p(x,y)$ in terms of an instance i of situation set P (see the previous subsection), the quantified expression may be restated as

$$\text{AxEyE}\langle P \ x \ y \rangle$$

or as

$$\text{AxEyEiE}\langle e \ i \ P \rangle \text{E}\langle \text{case1 } i \ x \rangle \text{E}\langle \text{case2 } i \ y \rangle .$$

The matrix space $M0$ encodes the situation

$$\langle P \ x \ y \rangle$$

or, equivalently, the conjunction of subsituations

$$\langle e \ i \ P \rangle \ \& \ \langle \text{case1 } i \ x \rangle \ \& \ \langle \text{case2 } i \ y \rangle .$$

The quantification spaces $E1$, $A1$, and $E0$ then add quantification information to this conjunction of situations.

Looking at the vista $(E1 \ A1 \ E0)$, nodes ' i ' and ' y ' and associated arc structures lie on existential space $E1$ and are therefore within the scope of (and dependent upon) all universal variables specified on universal spaces above $E1$ in the hierarchy. For this

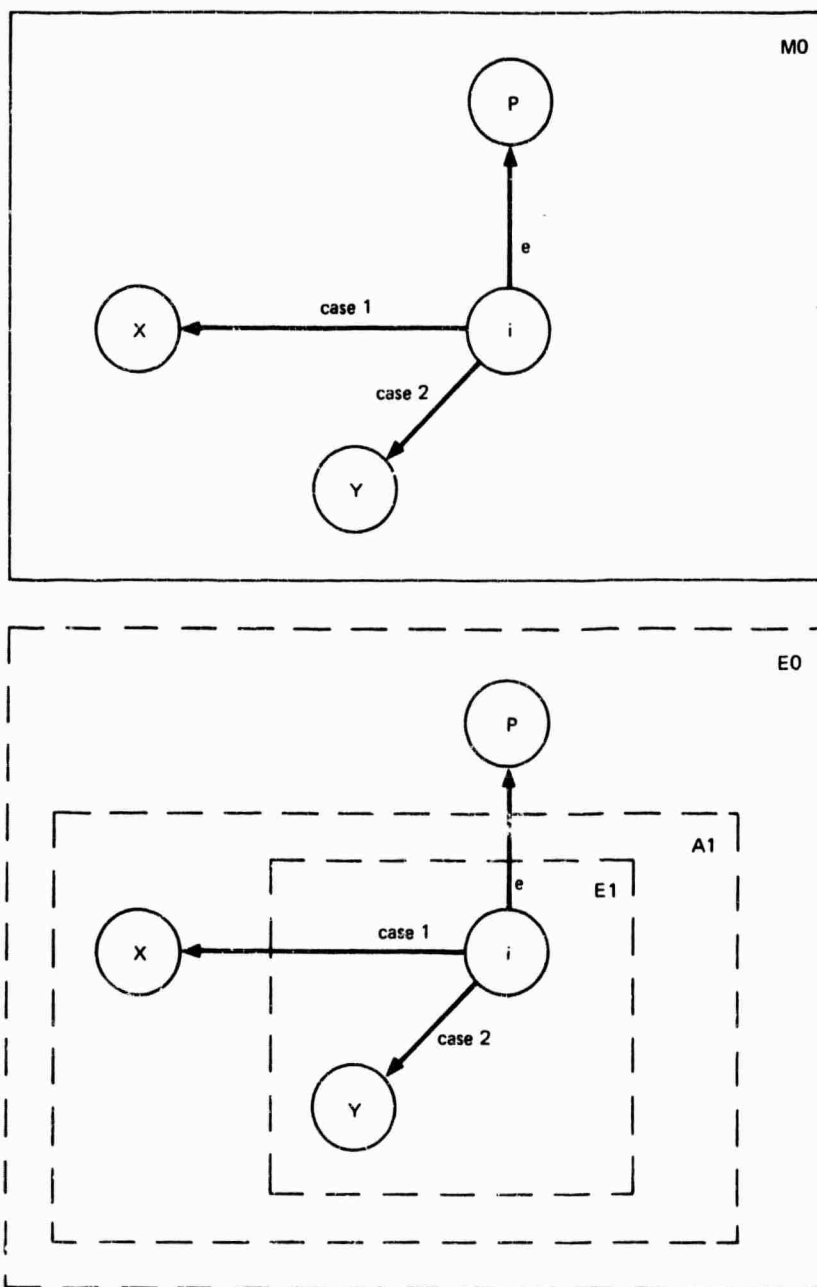


FIGURE V-19 THE ENCODING OF $AxEy [p(x,y)]$ BY THE ORTHOGONAL PARTITION METHOD

example, the only such universal space is A_1 , which specifies the universal variable x . Appearing on space E_0 , the space of top-level constants, is node 'P'. Hence, the network encodes P as a constant, x as a universal, and y and i as existentials within the scope of x . The various arcs that lie on E_1 may also be interpreted as existentials within the scope of x . Each such arc indicates an instance of a relationship that depends on x . For example, consider the e arc from 'i' to 'P'. Each x determines a new i and therefore a new instance of the element-of relation between that i and P . $\langle e \ i \ P \rangle$.

Even without considering more difficult examples or constructing formal proofs, it should be clear that quantified statements of arbitrary complexity may be encoded using this scheme. The scheme has a number of appealing features. Node and arc structures are not needed to explicitly encode quantifiers and scopes (as would be the case in any nonpartitioned network; see Shapiro, 1971). This leaves the nodes and arcs free to encode only matrix-type information, thus simplifying pattern-matching algorithms. The placing of quantification information in a separate partitioning is attractive in that it is analogous to "moving quantifiers to the left" in predicate calculus. Branching quantifiers are handled easily. Further, by consulting the vista of the quantification space upon which a node or arc lies, all variables of higher scope may be found easily.

However, the orthogonal partitioning approach to quantification adds new quantification spaces to the spaces encoding

logical connectives and therefore makes additional storage demands on the system. If the matrix spaces could themselves carry quantification information, then the quantification spaces could be eliminated, resulting in simpler networks, demanding less storage.

c. THE IMPLICIT EXISTENTIAL APPROACH TO QUANTIFICATION

The technique for encoding quantification that was actually employed in the SRI speech understanding system is called the "implicit existential (IE) approach." Using this technique, the spaces created to encode logical connectives serve double duty by also encoding quantification information.

The basic idea of the IE approach is to let each space used by the logical connectives implicitly carry an existential quantifier. Each connective space, it will be recalled, encodes a conjunctive situation incorporating a number of subsituations. When a connective space carries an existential quantification, it not only encodes this conjunctive situation but also asserts its existence.* Universal quantification is achieved indirectly in the IE approach through the use of the identity

$$Ax[p(x)] \Leftrightarrow \neg Ex[\neg p(x)] ,$$

which may be restated in English as "{for every x, p(x) is true} is equivalent with {there does not exist an x for which p(x) is false}."

* In the orthogonal approach, the connective space (the matrix space) encoded the conjunction situation, but information regarding its existence was encoded on quantification spaces.

That is, the IE approach transforms all universally quantified statements into statements involving only existentials and NEGATIONS.

As an example of the implicit existential approach to quantification, reconsider Figure V-12, assuming that each connective space implicitly carries an existential quantifier. Space S1 models the conjoined existences of such entities as General.Dynamics, BUILDINGS, and the like . It also models the existence of disjunction D and the membership of D in the set DISJUNCTIONS. Further, S1 encodes the existence of the two alternative disjuncts of D, S2, and S3. (Note carefully that the existence of alternatives does not imply the existence of situations specified in the alternatives.)

If one of the alternative disjuncts of D is accepted, then the information encoded by that alternative extends the model formed by S1. For example, if S2 is accepted, the model is extended to include the existence of entities matching the structure of S2. In particular, the acceptance of alternative S2 implies the acceptance of the existence of a building situation S whose agent is General.Dynamics and whose object is the Henry.L.Stimson.

The information encoded by the network of Figure V-12 is purely existential and therefore lends itself easily to the IE approach. For a general description of how universal quantification may be encoded in the IE approach, consider the abstract statement

$\text{AxEy}[p(x,y)]$,

which may be transformed as follows:

$AxEyE\langle P\ x\ y\rangle$
 $\sim\sim AxEyE\langle P\ x\ y\rangle$
 $\sim(\sim Ax)EyE\langle P\ x\ y\rangle$
 $\sim Ex[\sim\{EyE\langle P\ x\ y\rangle\}]$.

Since the last statement involves only existentials and logical connectives, it may be encoded directly by the IE approach, as shown in Figure V-20. The relationship between Figure V-20 and the last statement above should be clear. To understand better how the relationship $\{Ax[p(x)] \Leftrightarrow \sim Ex[\sim p(x)]\}$ is used in Figure V-20, note the parallelism between vista (S3 S2 S1) of that figure and vista (E1 A1 E0) of Figure V-19.

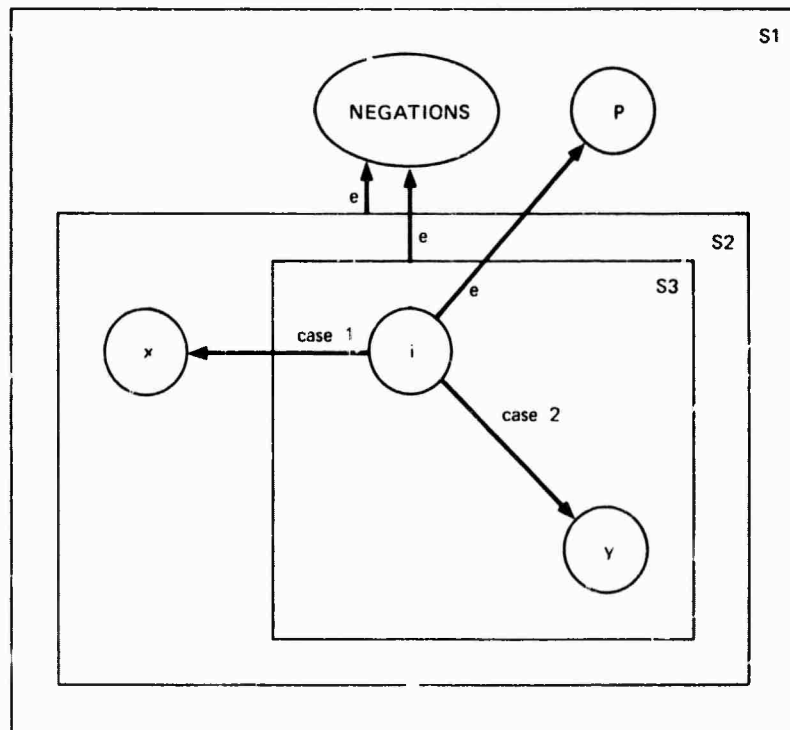


FIGURE V-20 AN ENCODING OF $AxEy\ [p(x,y)]$ BY THE IMPLICIT EXISTENTIAL METHOD

d. STREAMLINING THE IMPLICIT EXISTENTIAL APPROACH

Although the encoding of universal quantification as the negation of existential quantification is well-founded mathematically, it is not particularly intuitive and the resulting networks (such as Figure V-20) seem unnecessarily complex. To streamline the implicit existential approach, a shorthand notation based on the IMPLICATION connective may be adopted. This shorthand will both simplify the networks and increase their intuitive appeal.

Working toward this streamlining of universal quantification, consider the statement

$$Ax[p(x)] .$$

Although this statement looks extremely simple, it is the canonical form of all universals and may in fact encode quite complex information. The x , for example, may be a whole vector of variables, and p may carry additional universal and existential information. And, of course, this statement might be embedded in a more complex expression. The point is that if this statement can be analyzed and if an efficient means can be found for encoding it in a partitioned network, then similar techniques will apply to all universals.

For natural language systems (and probably for any system modeling a piece of the real world), the $p(x)$ in $Ax[p(x)]$ may almost always be restated as an implication of the form

$$p(x) = \{q(x) \Rightarrow r(x)\} .$$

This is because variables such as x are almost always quantified over some set (i.e., typed) when used in natural language.* The antecedent $q(x)$ of the implication serves to restrict the universally quantified variable, confining its range to some fixed set.

A given $q(x)$ or $r(x)$ might predicate the existence of other entities that interact with x . For example, it might be that

$$q(x) = \text{Ey}[u(x,y)]$$

and

$$r(x) = \text{Ez}[v(x,z)] .$$

If x , y , and z are thought of as (possibly empty) sets of variables and u and v are thought of as arbitrary formulas, then a statement of the form $\text{Ax}[p(x)]$ or, equivalently,

$$\text{Ax}[\{\text{Ey}[u(x,y)]\} \Rightarrow \{\text{Ez}[v(x,z)]\}]$$

is completely general. After making the transformation

$$\sim \text{Ex}[\sim \{\{\text{Ey}[u(x,y)]\} \Rightarrow \{\text{Ez}[v(x,z)]\}}]$$

the statement may be directly encoded as shown in the forbidding network of Figure V-21.

However, using a shorthand notation, the same information may be encoded by the simpler network of Figure V-22. Use of the

 * That is, since there is so very little that is true or interesting about everything, almost all universally quantified statements concern only the members of some subset of the universal set. For example, q may restrict x to only range over the set of COUNTRIES or over the set of SHIPS. Quantified English sentences encoding these restrictions might begin with "all countries ..." or "every ship ..." (Note that even the morphology of the word "everything" suggests the analysis "for all x , if x is a THING, then ...")

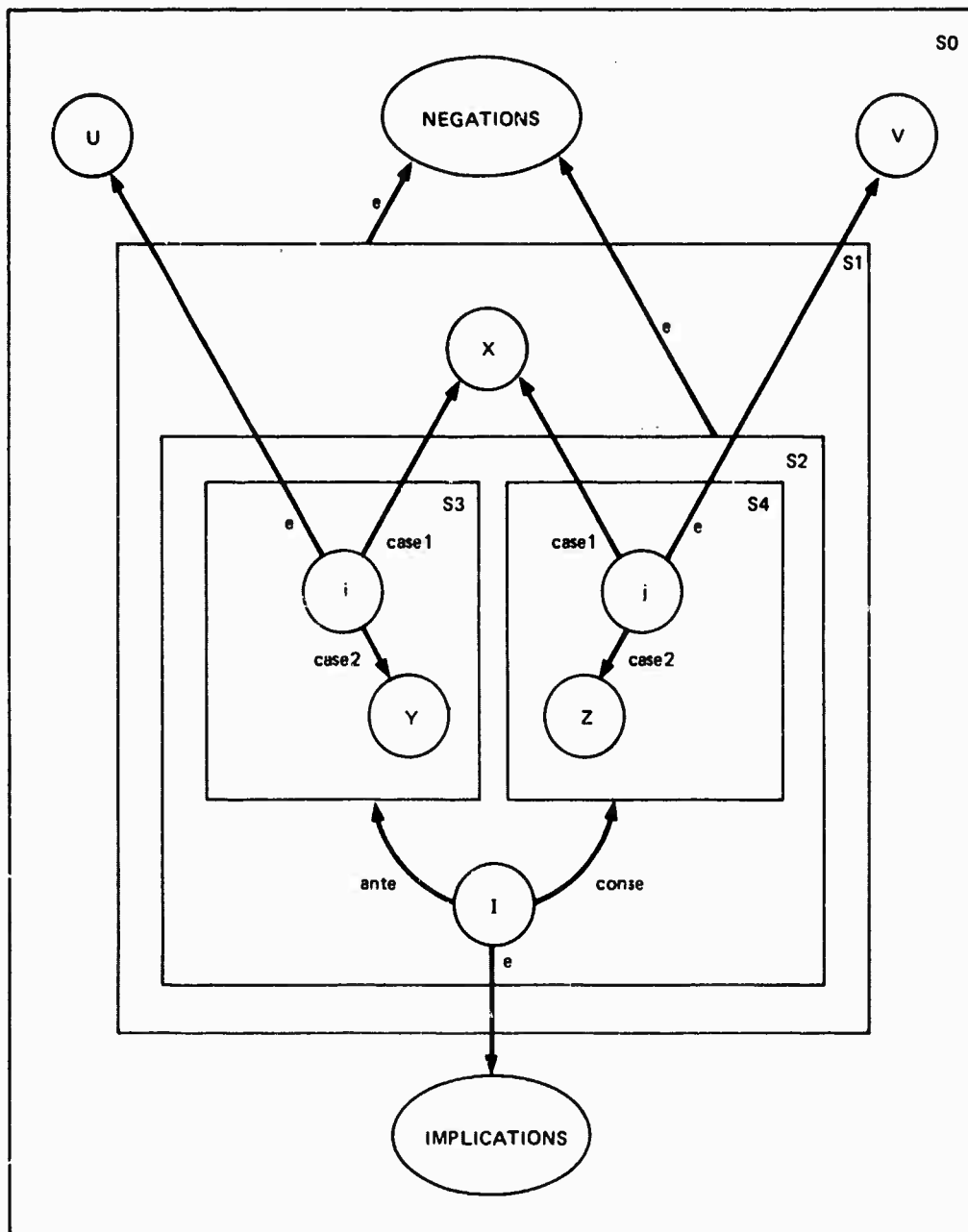


FIGURE V-21 AN ENCODING OF $Ax[\{Ey[u(x,y)]\} \Rightarrow \{Ez[v(x,z)]\}]$

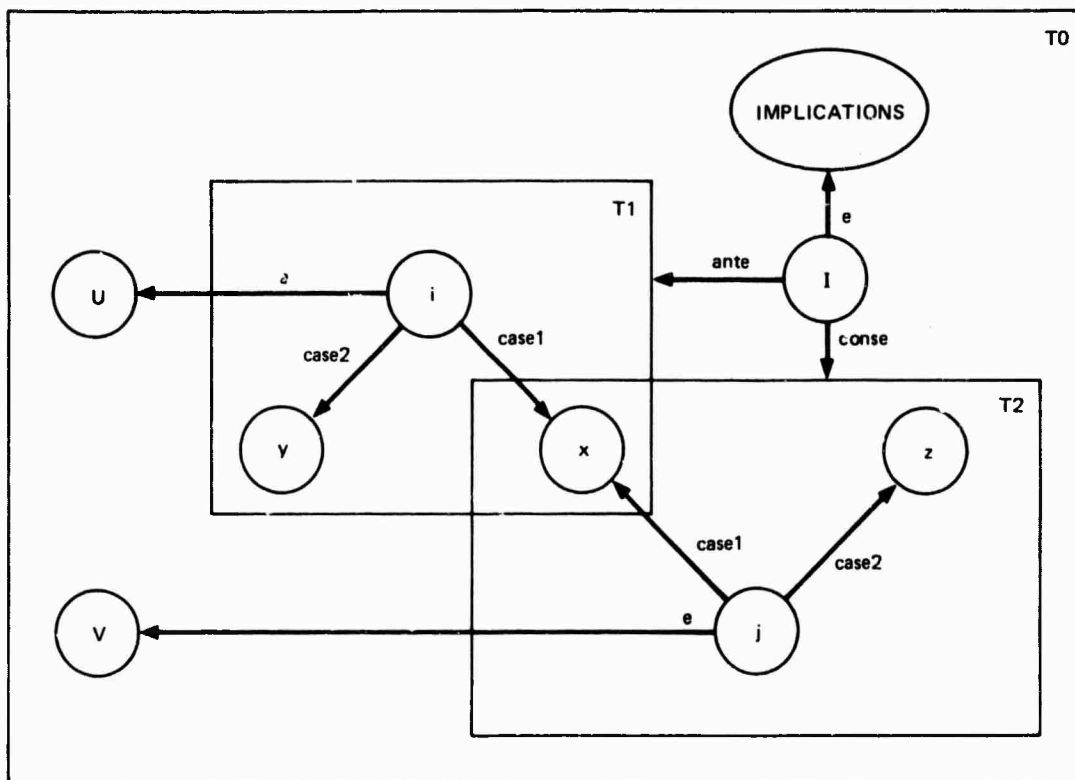


FIGURE V-22 A SHORTHAND ENCODING OF $Ax[\{Ey[u(x,y)]\} \Rightarrow \{Ez[v(x,z)]\}]$

shorthand is indicated by the overlapping of the ante and conse spaces of an implication. Whenever such an overlap occurs, the structures in the overlap are to be considered as being universally quantified and the remainder of the implication is to be considered within the scope of these universals.

[The deduction procedures that operate on partitioned semantic networks (see the Chapter XII on the deduction component)

actually consider all of the ante space structures, including the overlap, to be universally quantified. The justification for this arises from the following analysis:

$$\begin{aligned} & \text{Ax}[\{\text{Ey}[\text{u}(\text{x},\text{y})]\} \Rightarrow \{\text{Ez}[\text{v}(\text{x},\text{z})]\}] \\ & \text{Ax}[\sim\{\text{Ey}[\text{u}(\text{x},\text{y})]\} \vee \{\text{Ez}[\text{v}(\text{x},\text{z})]\}] \\ & \text{Ax}[\{\text{Ay}[\sim\text{u}(\text{x},\text{y})]\} \vee \{\text{Ez}[\text{v}(\text{x},\text{z})]\}] \\ & \text{AxAy}[\sim\text{u}(\text{x},\text{y}) \vee \{\text{Ez}[\text{v}(\text{x},\text{z})]\}] \\ & \text{AxAy}[\text{u}(\text{x},\text{y}) \Rightarrow \{\text{Ez}[\text{v}(\text{x},\text{z})]\}] \end{aligned}$$

The ability to include $\{\text{Ez}[\text{v}(\text{x},\text{z})]\}$ within the scope of y in the next to last step derives from the fact that $\{\text{Ez}[\text{v}(\text{x},\text{z})]\}$ is independent of y .]

The correspondences between the nets of Figure V-21 and Figure V-22 are as follows: Space S1 corresponds to the overlap of T1 and T2. S3 corresponds to T1 less the overlap. S4 corresponds to T2 less the overlap. Note that when the negations are ignored, the view of the network from the vantage of S3 (or, alternatively, S4) is identical to the view from T1 (T2). But rather than inherit a view of node 'X' as in vista (S3 S2 S1 S0), the view from vista (T1 T0) includes 'X' because 'X' appears directly in T1.

Since spaces S3 and S4 and the implication are within a double negative in Figure V-21, their corresponding structures in Figure V-22 are allowed to appear at the top level. It is this elimination of a double negative that results in the simplification of structure.

Intuitively, the implication structure of Figure V-22 may be interpreted as meaning

for the existence of any entities matching the structure of

space T1, there will exist entities to match the structure of T2 as well, with the match for structures in the overlap being the same for both T1 and T2.

e. EXAMPLES OF THE OVERLAP SHORTHAND

Since the shorthand presented above provides the principal means for encoding universal quantification in the SRI speech understanding system, this subsection provides four examples that further expound and clarify its use.

As the first example, consider

"Every submarine is owned by a/some country."

which may be formalized as

AxEc[member(x,SUBMARINES) =>
{member(c,COUNTRIES) & owns(c,x)}} .

The network encoding of this statement appears in Figure V-23. The universal quantification of x is indicated by its appearance in the ante/conse overlap. The dependence of country c on submarine x is indicated by the appearance of c in the conse space S3, within the (shorthand implied) scope of x.

The example of Figure V-23 may be contrasted with the example of Figure V-24, which encodes

"Every Lafayette (class sub) is owned by the U.S."

or

Ax[member(x,LAFAYETTES) => owns(The.U.S.,x)] .

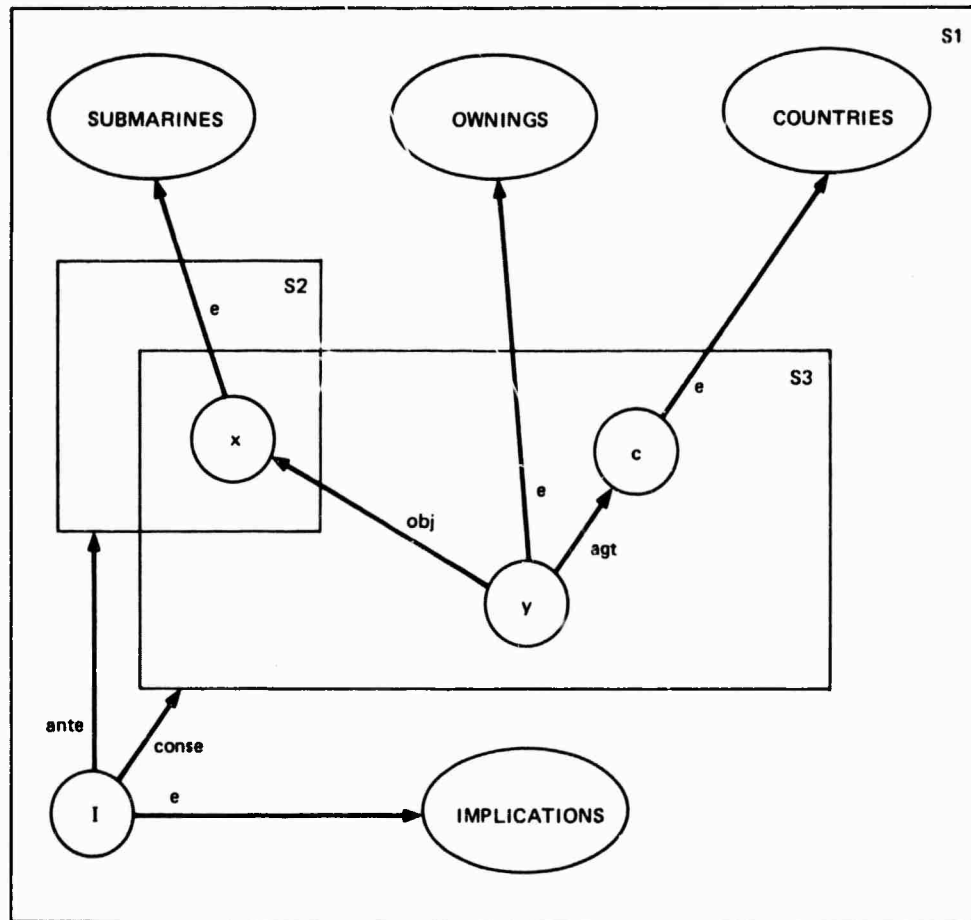


FIGURE V-23 EVERY SUBMARINE IS OWNED BY SOME COUNTRY

Unlike the *c* above, the agent of these ownings is not within the scope of *x* and so lies on the higher space *S1*. The quantification expression might well have begun "THERE-EXISTS The.U.S and FOR-ALL *x* ..."

To show the nesting of scopes and the alternation of universal and existential quantifiers, consider the statement

"All the ships in any given class have the same length."

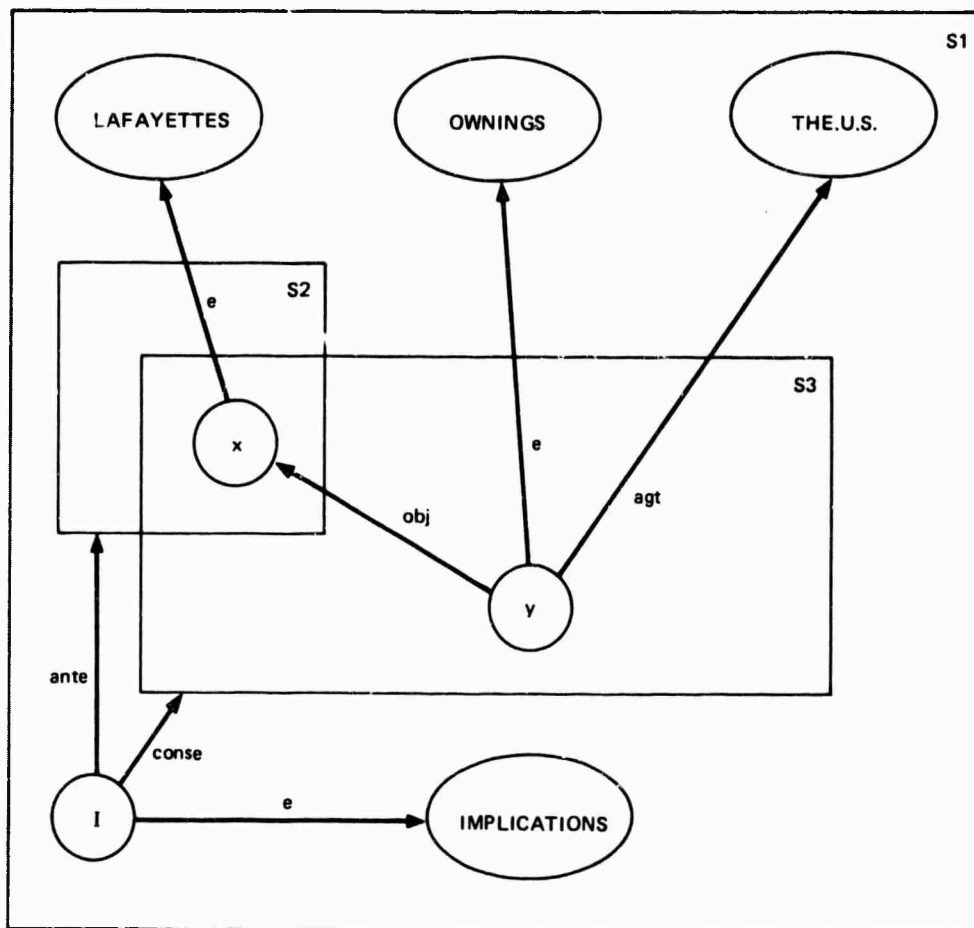


FIGURE V-24 EVERY LAFAYETTE IS OWNED BY THE U.S.

If a class of ships is thought of as a set (whose members belong to the class), then the statement may be formalized as

$$\text{AcElAs}[\text{member}(c, \text{CLASSES}) \Rightarrow \{\text{member}(l, \text{LENGTHS}) \& \{\text{member}(s, c) \Rightarrow \text{has.length}(s, l)\}\}]$$

or as

$$\text{Ac}[\text{member}(c, \text{CLASSES}) \Rightarrow \{\text{El}[\text{member}(l, \text{LENGTHS}) \& \text{As}[\text{member}(s, c) \Rightarrow \text{has.length}(s, l)]]\}] .$$

The network of Figure V-25 encodes this information, closely paralleling the last formal expression. Note that the universal variables *c* and *s* lie in overlaps and that *l* lies in *S3*, within the scope of *c*.

Since all the antecedents presented thus far have encoded only simple membership relationships, the fourth example, encoded by the net of Figure V-26, has a more complex ante. The example statement is

"All ships built by General.Dynamics belong to the U.S."

or

$Ax[\text{member}(x, \text{SHIPS}) \ \& \ \text{built}(\text{General.Dynamics}, x) \\ \Rightarrow \text{owns}(\text{The.U.S.}, x)]$.

In this example, the set restricting the values of *x* is not explicitly encoded in the network. Nevertheless, the antecedent of the implication restricts *x* to be taken from the set of ships that were built by General.Dynamics.

f. QUANTIFICATION IN THE HIERARCHICAL TAXONOMY

The use of *s* and *ds* arcs for creating a hierarchical taxonomy of objects was presented as a basic network concept. The taxonomic information alone provides answers to element/set/subset questions such as

"Is the Henry.L.Stimson a ship?"

"Are destroyers ships?"

"Is a submarine a destroyer?"

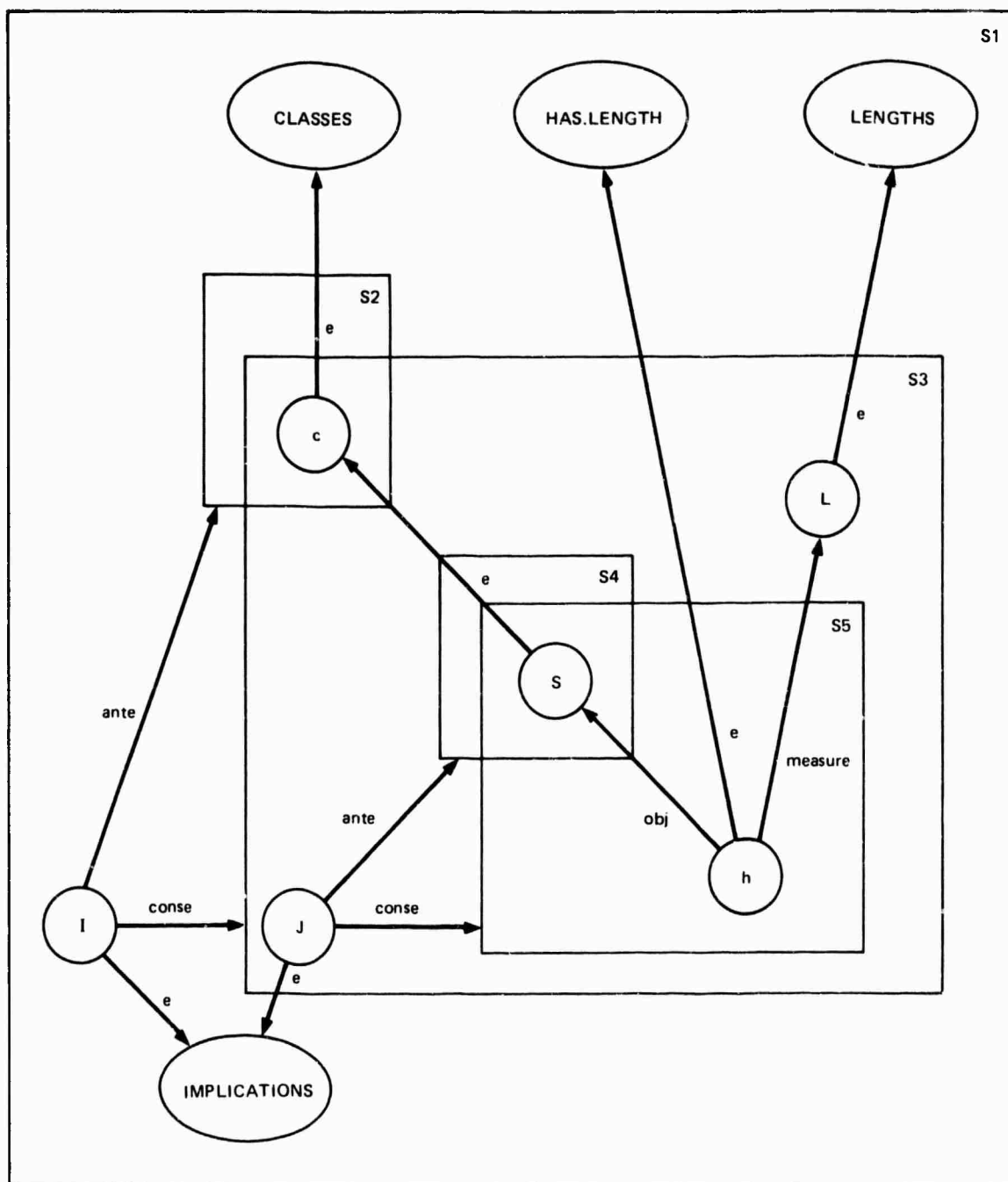


FIGURE V-25 ALL THE SHIPS IN ANY GIVEN CLASS HAVE THE SAME LENGTH

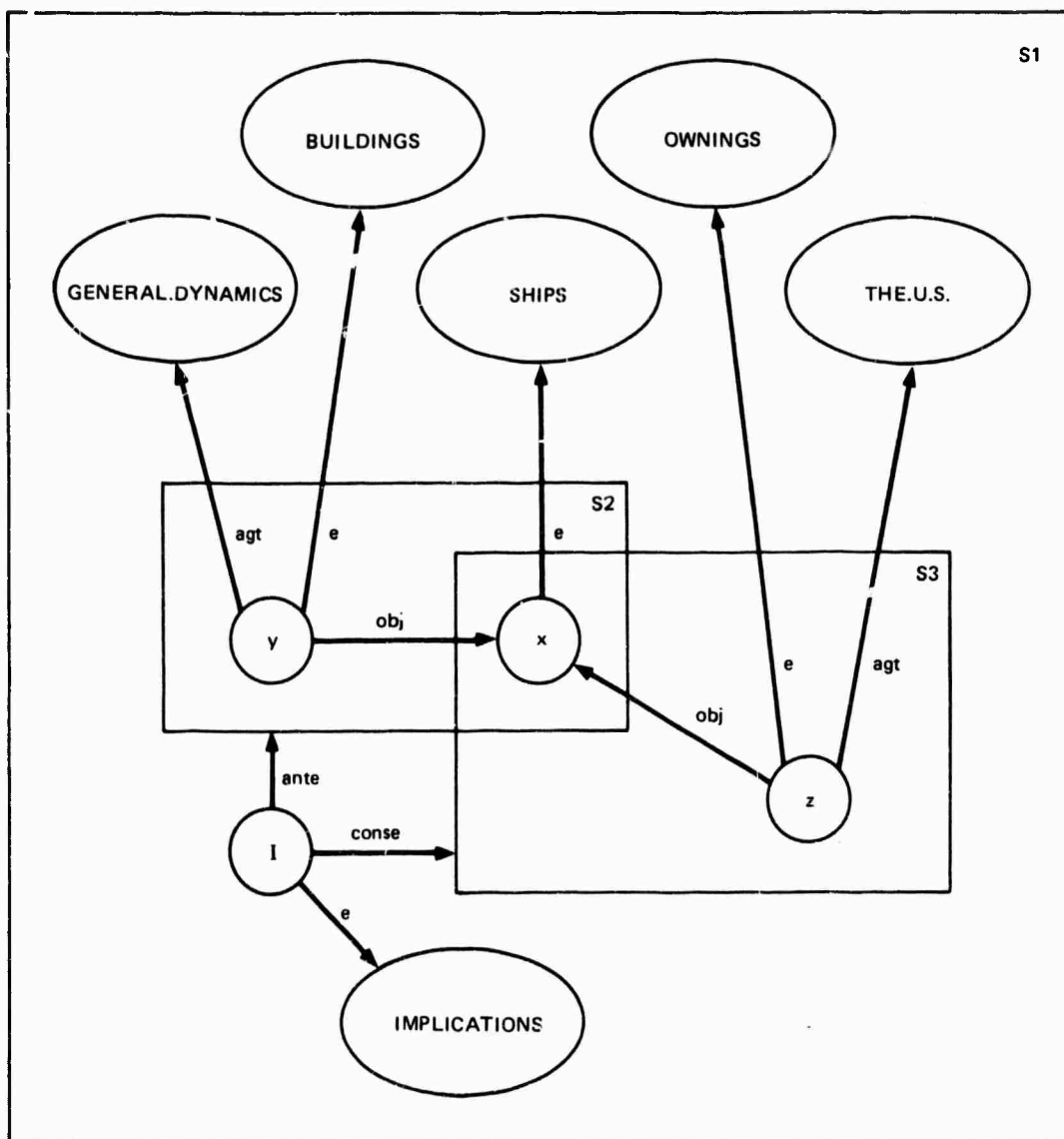


FIGURE V-26 ALL SHIPS BUILT BY GENERAL.DYNAMICS BELONG TO THE U.S.

However, taxonomic information takes on added significance when the various sets of the taxonomy are associated with general rules (i.e., quantified statements). For example, if the rule

$$\text{Ax}[\text{member}(x,S) \Rightarrow p(x)]$$

is included in the system, then knowing that some individual i is a member of set S is enough to establish that i has property p . Furthermore (and this is the important point), there is no need to explicitly record the fact that individual i has property p since it may be easily derived. If set S has n members, the encoding of the one general rule and the various set memberships (many of which may be derived in the taxonomy through chains of s arcs) saves n reencodings of p for the various members.

In the domain of the SRI speech understanding system, it happens that the ships that belong to a particular class have many properties in common. These common properties have been encoded in general rules to avoid replication of data. As an example of one such general rule, reconsider the network of Figure V-24, which states that all ships of the Lafayette class (all members of LAFAYETTES) have the property of being owned by The.U.S.

g. DELINEATIONS

By indicating some of the common properties p of members of a set S , a universally quantified statement serves to partially bound S . That is, by stating that all members of S have property p , a general

rule indicates that ONLY individuals having property p are in S. Thus, the general rule provides an indication of a limitation on the membership of S. Formally, this limitation arises as a consequence of the fact that

$$\{Ax[\text{member}(x,S) \Rightarrow p(x)]\} \Leftrightarrow \{Ax[\sim p(x) \Rightarrow \sim \text{member}(x,S)]\} .$$

For purposes of understanding natural language inputs, general rules serving to limit the membership of situation sets are very important. In particular, it is useful for each situation set to have a general rule, called the set "delineation" rule, that names and restricts the participants of situations in the set. For example, the delineation rule of the set OWNINGS is shown in Figure V-27. This general rule indicates that all owning situations have an agt, obj, start-time, and end-time. Further, the agt must be a member of LECAL.PERSONS, the obj must be (in this system) a member of PHYSOBSJS, and the start-time and end-time must be elements of TIMES. More complex restrictions could also be added. For example, the start-time could be restricted to precede the end-time.

By using delineation rules, the semantic composition rules (which are discussed in detail in the Chapter VII) are able to reject certain anomalous combinations of phrases that nevertheless meet syntactic and acoustic criteria for being joined. For example, if various indicators suggest the hypothesis that the input utterance mentions an ownership situation in which the role of obj (not agt) is played by a country, then the delineation of OWNINGS may be used to

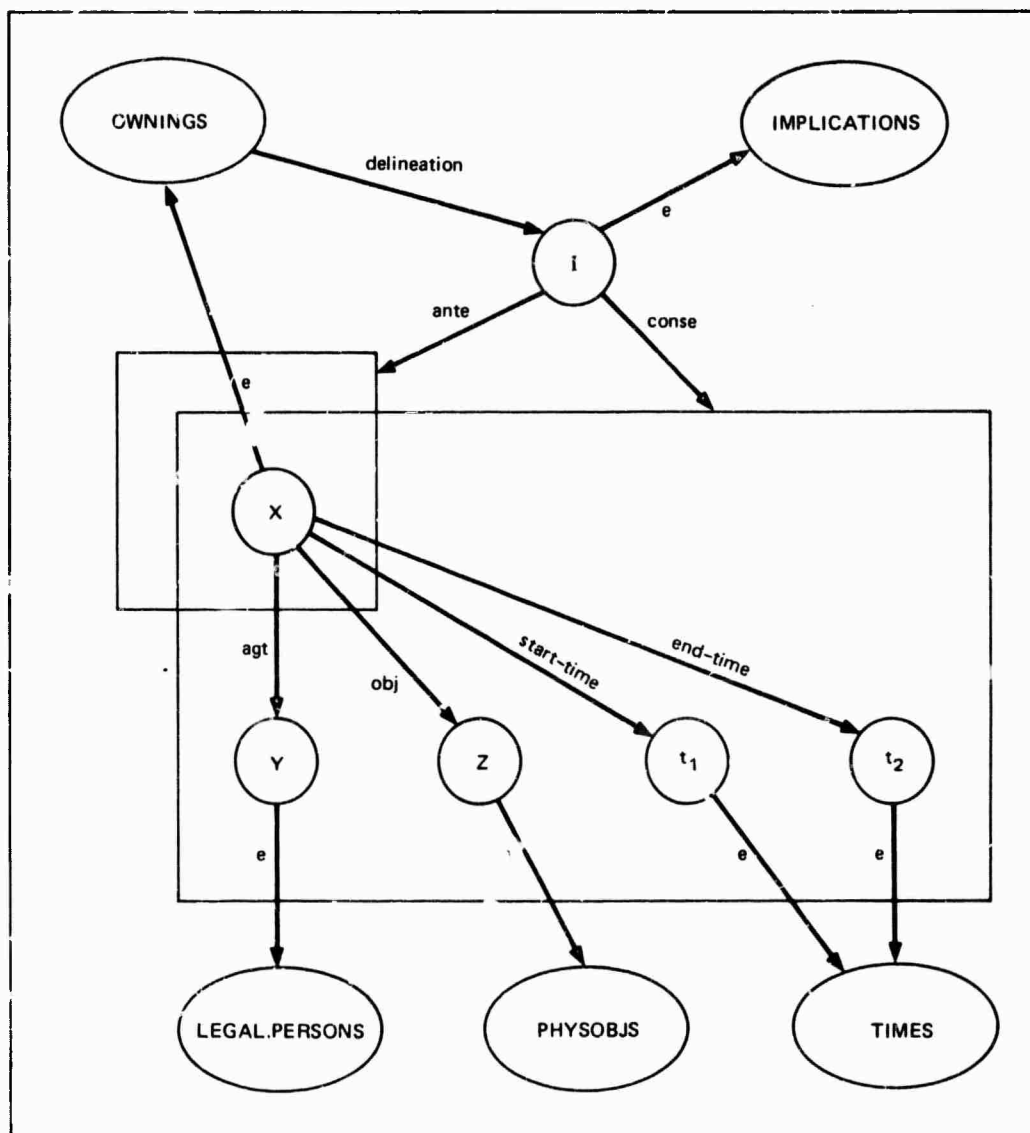


FIGURE V-27 THE DELINEATION THEOREM OF OWNINGS

reject the hypothesis on the grounds that the role of obj may be filled only by elements of PhYSOBS. (The fact that no country is a physical object follows immediately from the taxonomy.)

Since delineations are a common commodity in the SRI speech understanding system, their structure is sometimes abbreviated in figures to simplify the notation. Such an abbreviation is shown in the drawing of Figure V-28, which is intended to convey the same information as the drawing of Figure V-27. Unlike the implication shorthand discussed earlier, the delineation abbreviation is used only in pictures and is not actually reflected in the internal computer structures.

3 OTHER HIGHER-ORDER STRUCTURES

In addition to encoding quantification and logical connectives, partitioning may be used in the encoding of other higher-order structures as well. The need for additional higher-order structures is, of course, dependent upon the task that the system is to perform and the domain to be modeled.

For example, if a system is to deal with beliefs, then some second-order constructs for encoding beliefs must be devised. One possibility for encoding a belief system has already been presented in Figure V-10 (and is being pursued by Cohen and Perrault, 1976).

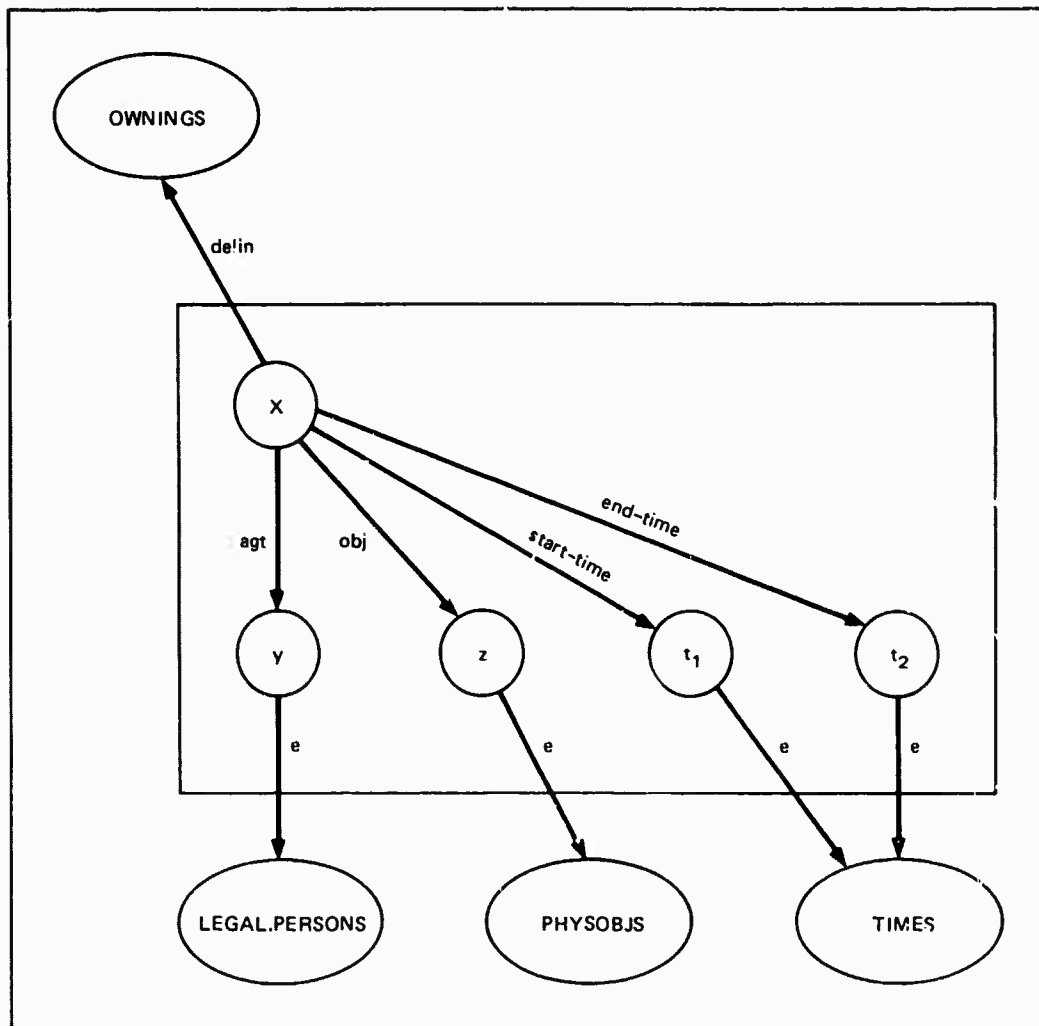


FIGURE V-28 ABBREVIATED DELINEATION OF OWNINGS

If a system were to be able to discuss the semantics of natural language, higher-order predicates would be needed to show the relationships between sentences (and parts of sentences) and their semantic interpretations. This might be done as in Figure V-29, which shows an interpretation situation X existing between the syntactic entity

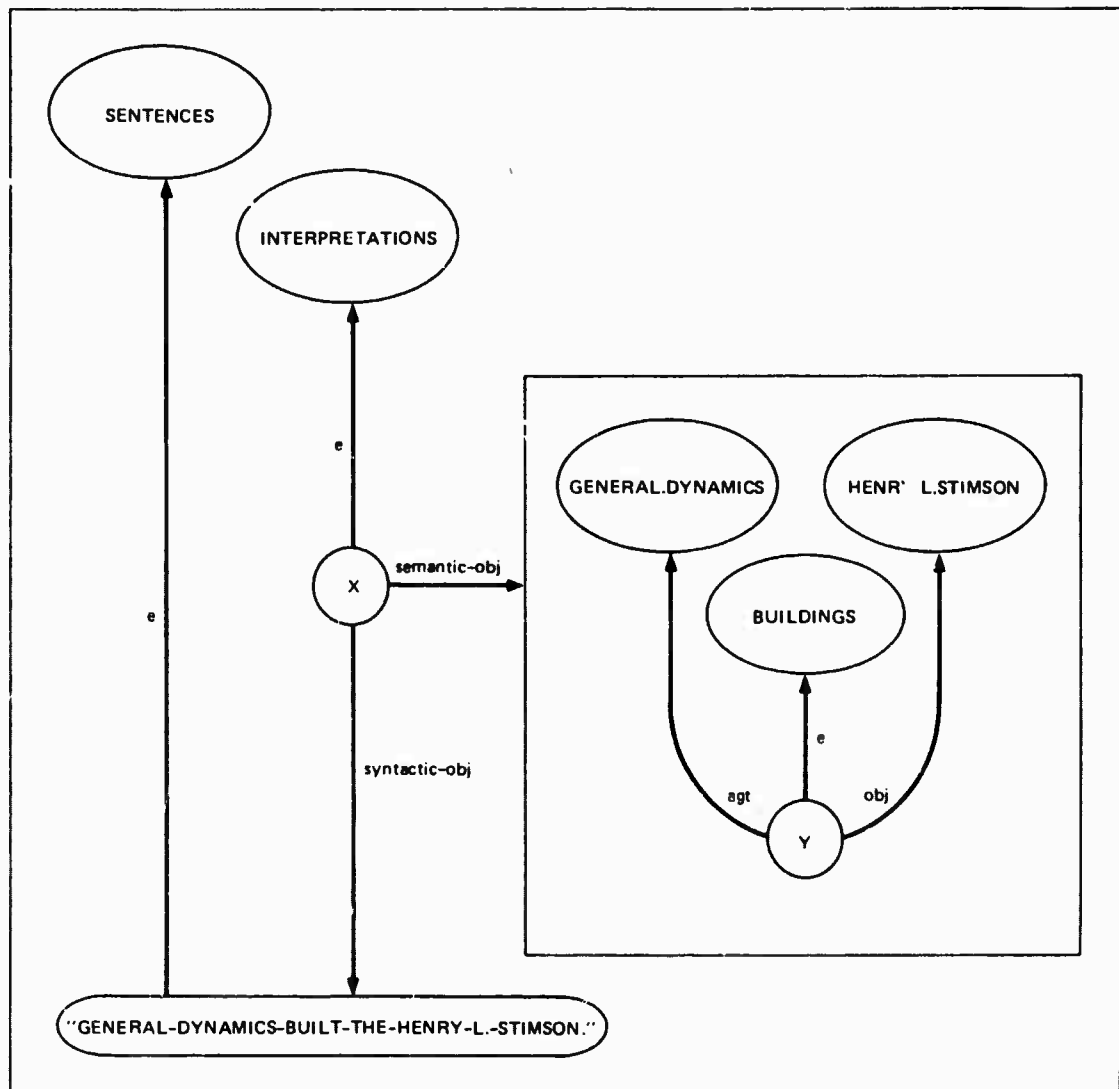


FIGURE V-29 RELATING A SENTENCE TO ITS MEANING

"General.Dynamics built the Henry.L.Stimson."
and its semantic interpretation in the net.

In the SRI speech understanding system, the task to be performed was that of understanding and responding to spoken inputs, most of which turned out to be questions. Hence, the principal need for new higher-order structures was for encoding queries.

a. REPRESENTING YES/NO QUERIES

Questions may be regarded as requests (or commands) for the delivery of information, with each such request carrying an indication of the nature of the sought data. YES/NO questions seek information regarding the validity of a proposition and supply the proposition itself as an indicator. For example, the question

"Did General.Dynamics build the Henry.L.Stimson?"
may be restated as the request

"Tell me if the statement 'General.Dynamics built the
Henry.L.Stimson' is true."

Figure V-30 shows a network encoding of this query. The query itself is represented by node 'Q'. Q is shown to be an element of REQUESTS.YN, the set of all requests for YES/NO-type information. Each such request has one component part, the proposition (prop) whose validity is being sought. Hence, the prop arc from 'Q' points to space S2, which encodes the proposition

built(General.Dynamics, Henry.L.Stimson).

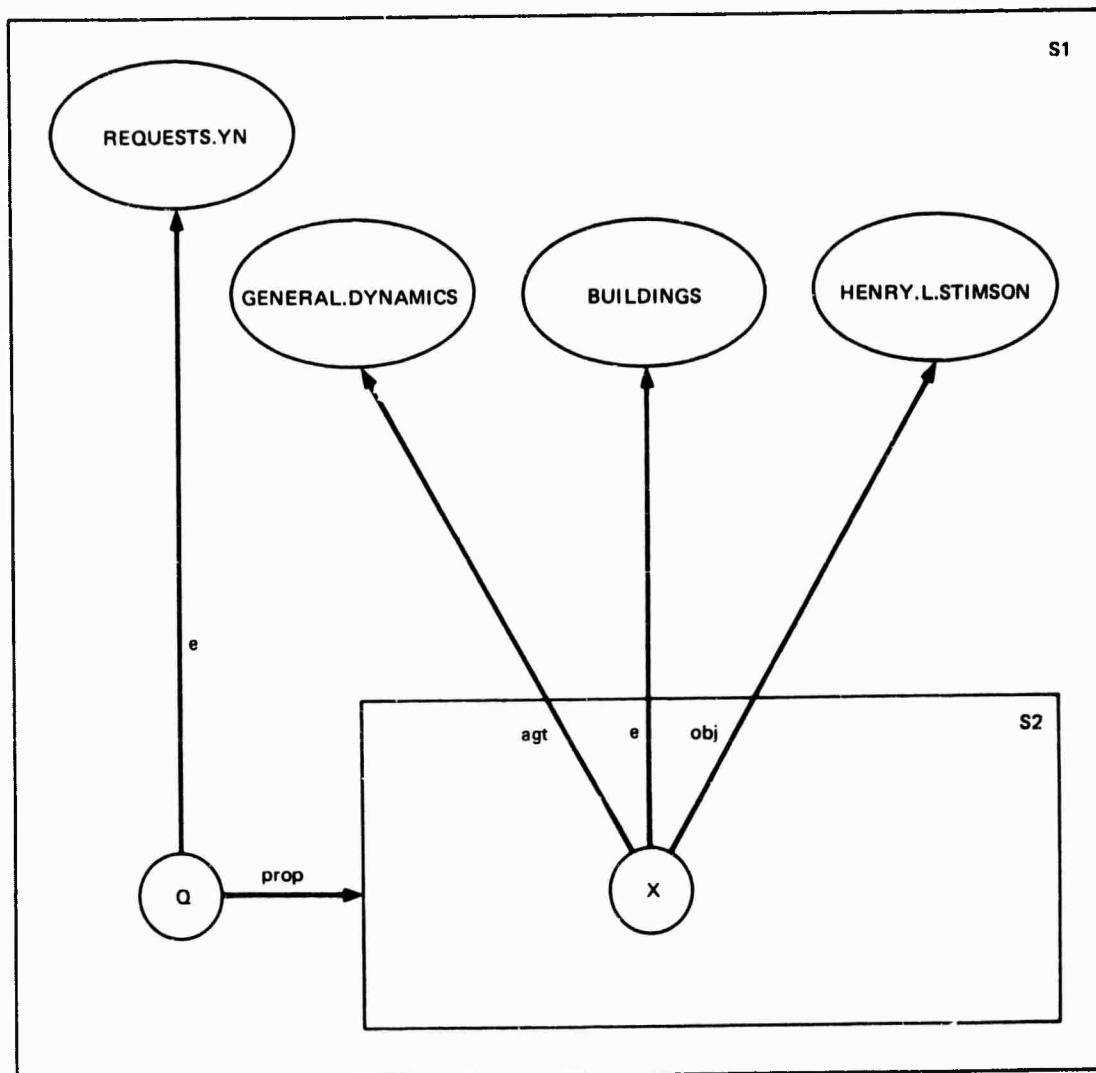


FIGURE V-30 DID GENERAL.DYNAMICS BUILD THE HENRY.L.STIMSON?

Figure V-31 shows the network encoding of the question

"Did General.Dynamics build all U.S. destroyers?"
whose associated proposition contains a universal quantification.

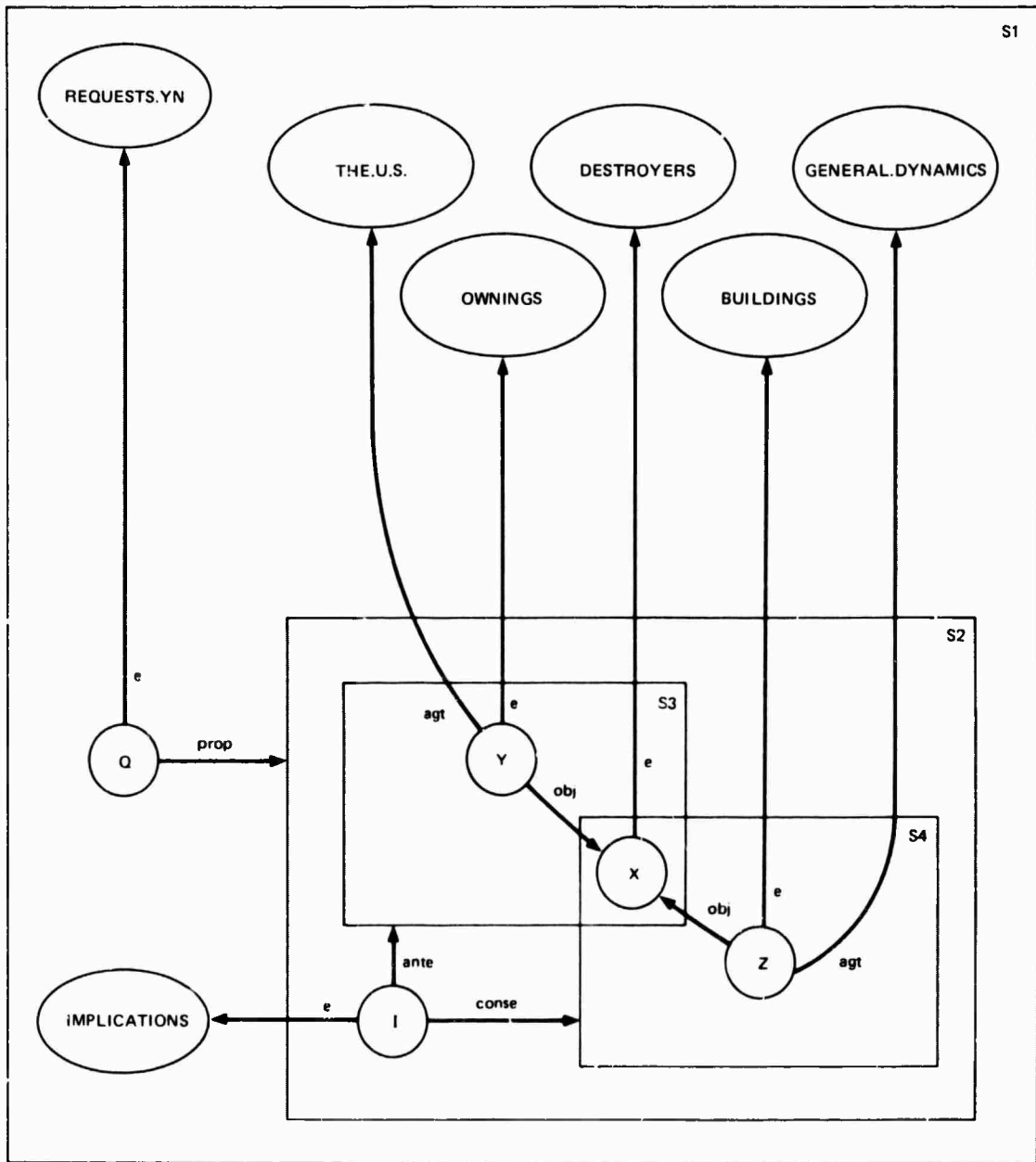


FIGURE V-31 DID GENERAL.DYNAMICS BUILD ALL U.S. DESTROYERS?

(To save storage, it would be possible to eliminate the 'Q' node and the prop arc by encoding the proposition itself as an element of REQUESTS.YN.)

b. REPRESENTING WH QUERIES

Like YES/NO queries, WH queries (queries concerning WHO, WHAT, WHICH, WHERE, HOW-MANY) are requests for information. But rather than simply querying the truth or falsity of a proposition, a WH query seeks to determine what bindings of existential variables will make a proposition true. For example, the WH query

"Who built the Henry.L.Stimson?"

is associated with the proposition

Ex[built(x, Henry.L.Stimson)] .

Working under the assumption that "some x built the Henry.L.Stimson", the query seeks bindings for existential variable x. A restatement of the query as a request might go something like this:

"Tell me what bindings of variable x will make the statement 'x built the Henry.L.Stimson' be true."

Figure V-32 shows a network encoding of this query/request. Q is the query itself, an element of the set REQUESTS.WH, the set of all requests for WH-type information. Each such request has two component parts, a proposition (prop) and an index. The index indicates those existential variables appearing in the proposition whose bindings are sought as answers to the query. The index is encoded

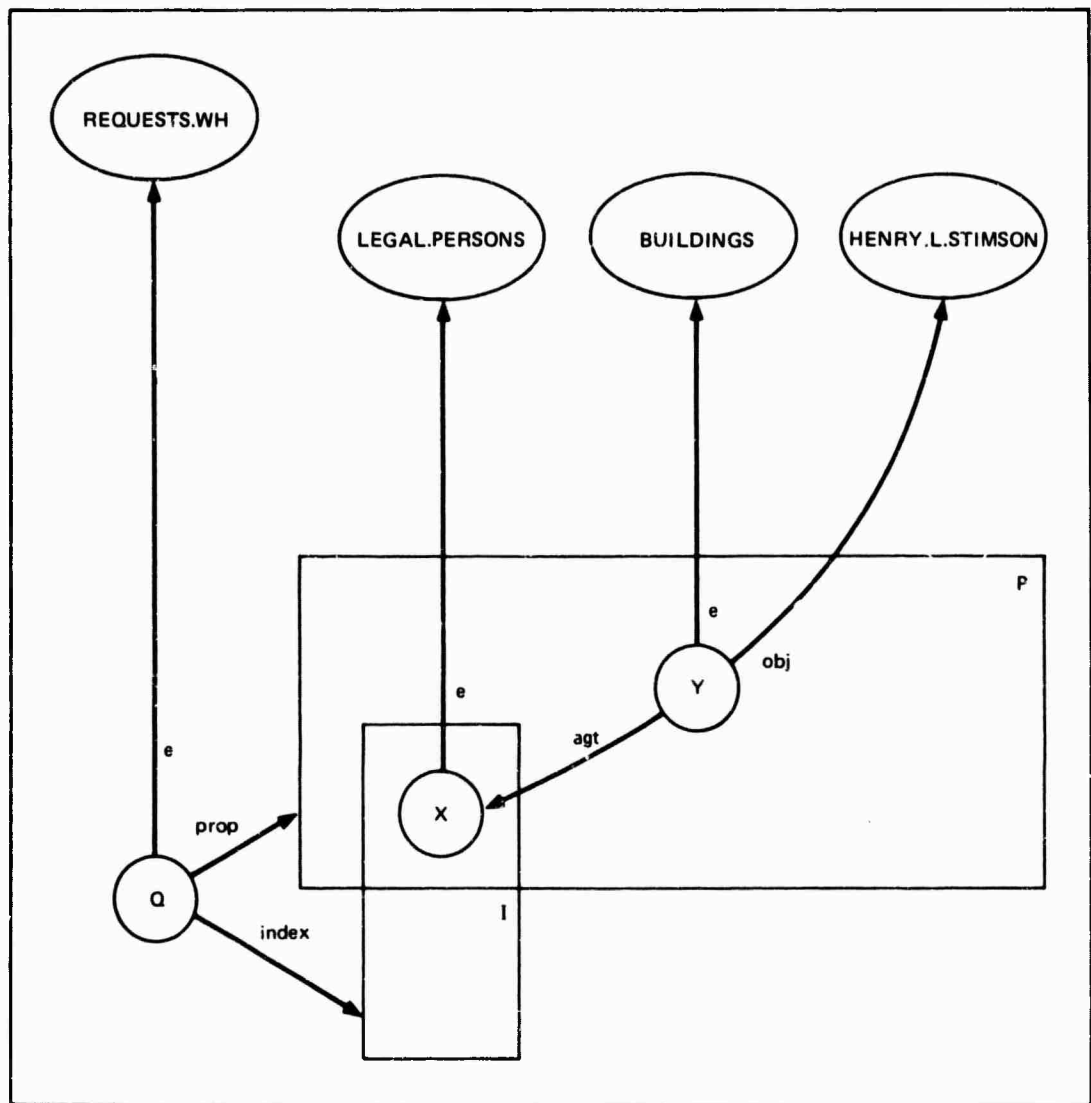


FIGURE V-32 WHO BUILT THE HENRY.L.STIMSON?

as a space that overlaps the space(s) representing the proposition and that includes only those nodes representing sought-after existentials.

The translation of the word "who" in this example is worth noting. The "who", of course, corresponds to node 'x'. But note that x is shown to be an element of `LEGAL.PERSONS`, the set of all humans, countries, companies, and the like. Thus, "who" translates roughly into "what member of the set of legal persons". Note also that "who" may be either singular or plural. A conscious decision has been made to treat these cases identically and to treat each WH request as a request for a set of answers. This set, which may contain zero, one, or multiple members, is generated one item at a time by the deduction component of the speech understanding system. (The deduction component, described in Chapter XII, sets up a generator-type coroutine. The first pulse of this generator produces one answer. More answers, if any, may be found by pulsing again.) A distinction between the plural and singular cases could easily be made. For example, elements of `REQUESTS.WH.SG` might request one binding of existential variables. Elements of `REQUESTS.WH.PL` might request all bindings of the existentials.

[Should storage become tight, it would be possible to eliminate the 'Q' node and prop arc (just as in YES/NO) by encoding the proposition itself as an element of `REQUESTS.WH` and by using the proposition space, in its role as supernode, as the tail of the index arc.]

The network of Figure V-33 encodes one reading of the English question

"Who built every destroyer?"

In particular, this network encodes the reading that assumes there exist one or more y , each of whom has individually built all of the destroyers, and that it is the identity of these y that is sought. (The uniqueness of the builder of an object is another matter which is not addressed in this example. If it were, the interpretation would then be that there is a unique y who built all of the destroyers.) Formally, a binding is sought for the y of

$\text{E}y \text{Ax} [\text{member}(x, \text{DESTROYERS}) \Rightarrow \text{built}(y, x)]$.

(If the interpretation placed upon this question proves troublesome, consider "who answered every question correctly?")

The network of Figure V-33 may be contrasted with the network of Figure V-34, which encodes one reading of the English question

"Who built each destroyer?"

The network of Figure V-34 places the request for information within the scope of a universal variable x that ranges over the set of DESTROYERS. That is, for each destroyer x , there is a new request for information. In particular, for each x , it is assumed that there exists a y who built x and the identity of this y is sought. An appropriate answer to this query would be something like "the x_1 was built by y_1 , the x_2 by y_2 , ..."

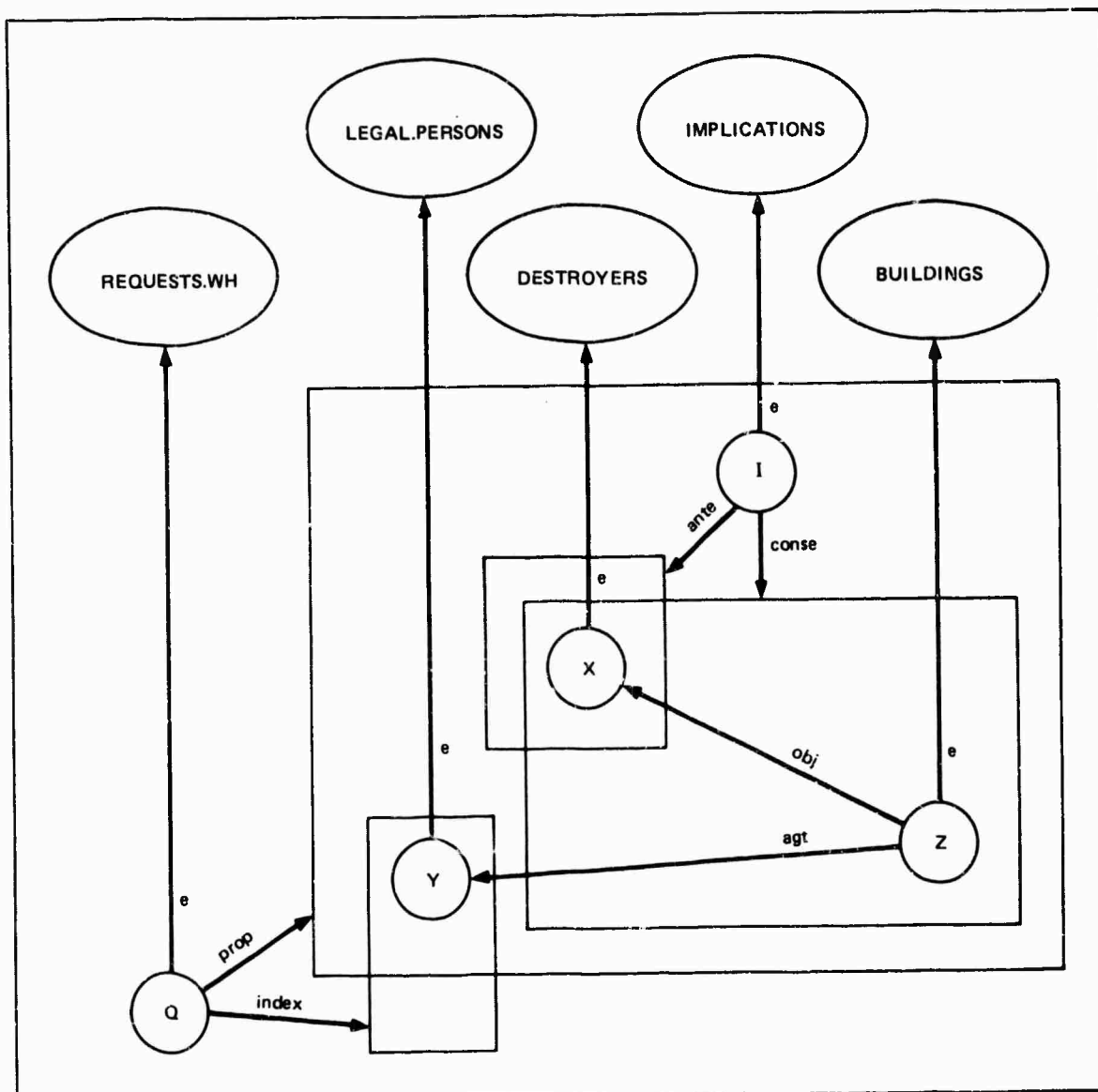


FIGURE V-33 WHO BUILT EVERY DESTROYER?

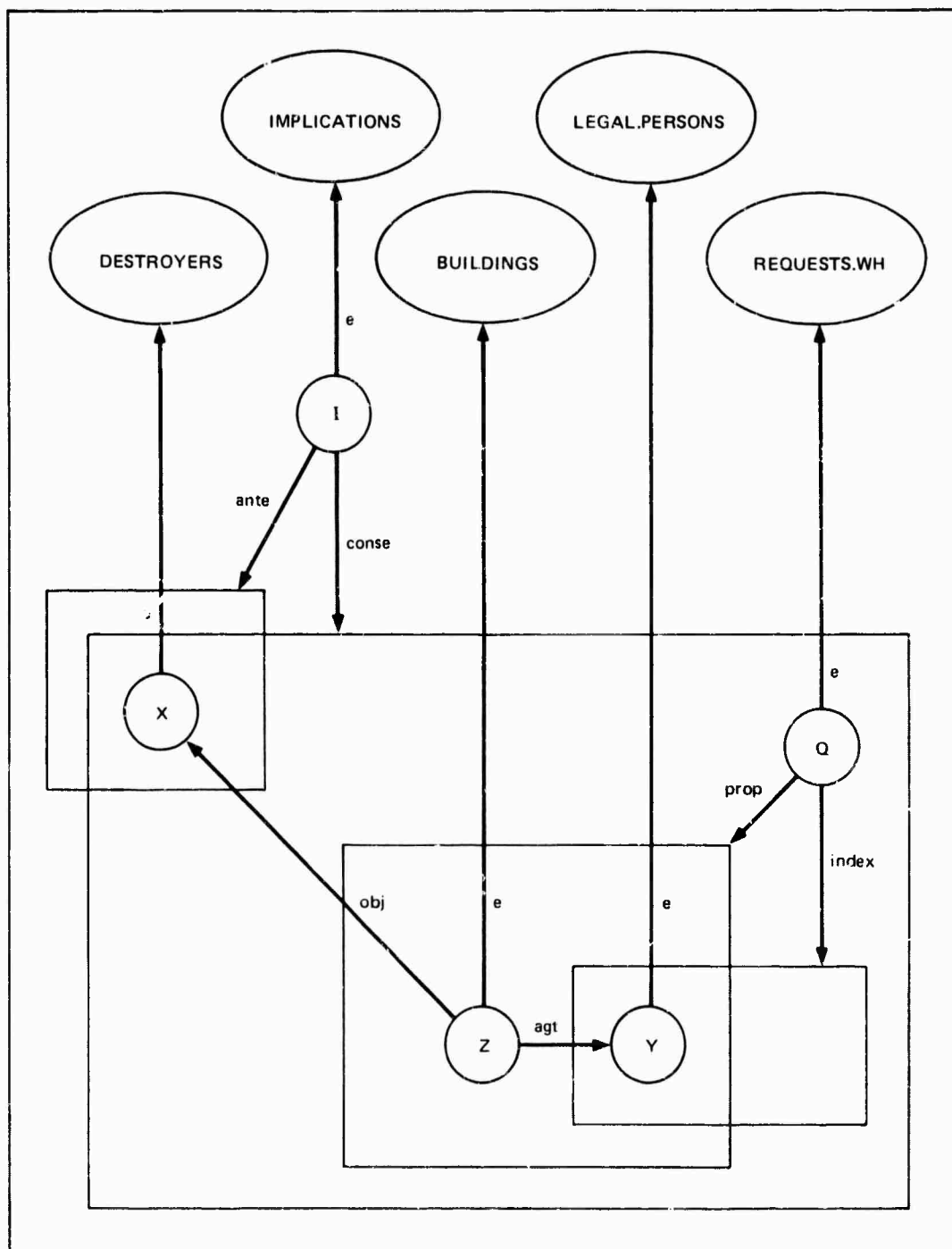


FIGURE V-34 WHO BUILT EACH DESTROYER?

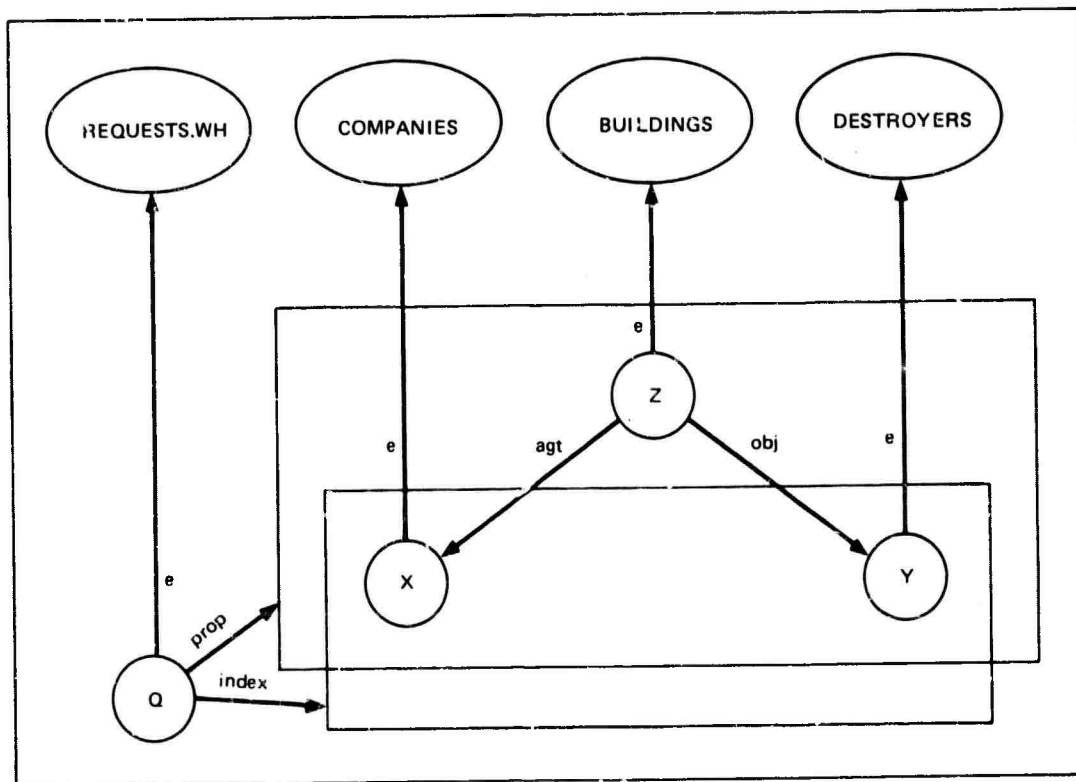


FIGURE V-35 WHAT COMPANIES BUILT WHAT DESTROYERS?

Figure V-35 shows the network encoding of the multiple WH question

"What companies built what destroyers?"

The underlying proposition is

$\text{ExEy}[\text{member}(x, \text{COMPANIES}) \ \& \ \text{member}(y, \text{DESTROYERS}) \ \& \ \text{built}(x, y)]$.

Bindings (all sets of bindings) are sought pairwise for both x and y .

Hence, both node ' x ' and node ' y ' lie on the index space.

c. REPRESENTING HOW-MANY QUERIES

A HOW-MANY question may be regarded as a special type of WH question that queries the cardinality of a set. For example, the HOW-MANY question

"How many ships did General.Dynamics build?"

may be rephrased as the WH question

"What is the cardinality (n) of the set (z) of ships (x) that were built by General.Dynamics?"

Formally, the associated proposition is

$$\text{EzEn}[\text{subset}(z, \text{SHIPS}) \ \& \ \text{member}(n, \text{NUMBERS}) \ \& \ \text{cardinality}(z, n) \ \& \\ \text{Ax}[\text{member}(x, z) \ \<=> \ \{\text{member}(x, \text{SHIPS}) \ \& \ \text{built}(\text{General.Dynamics}, x)\}]]$$

It is the binding of n that is sought by the query. Figure V-36 shows the network encoding that parallels this analysis.

One of the interesting features of the proposition in this question is the specification of set z. Set z is defined by stating a necessary and sufficient condition for set membership. Namely, "x is a member of z IF AND ONLY IF z is a ship and was built by General.Dynamics." This necessary and sufficient condition is encoded both in the predicate calculus formula and in the network as a two-way implication. Note in particular that implications I and J of the network make dual use of spaces S4 and S5.

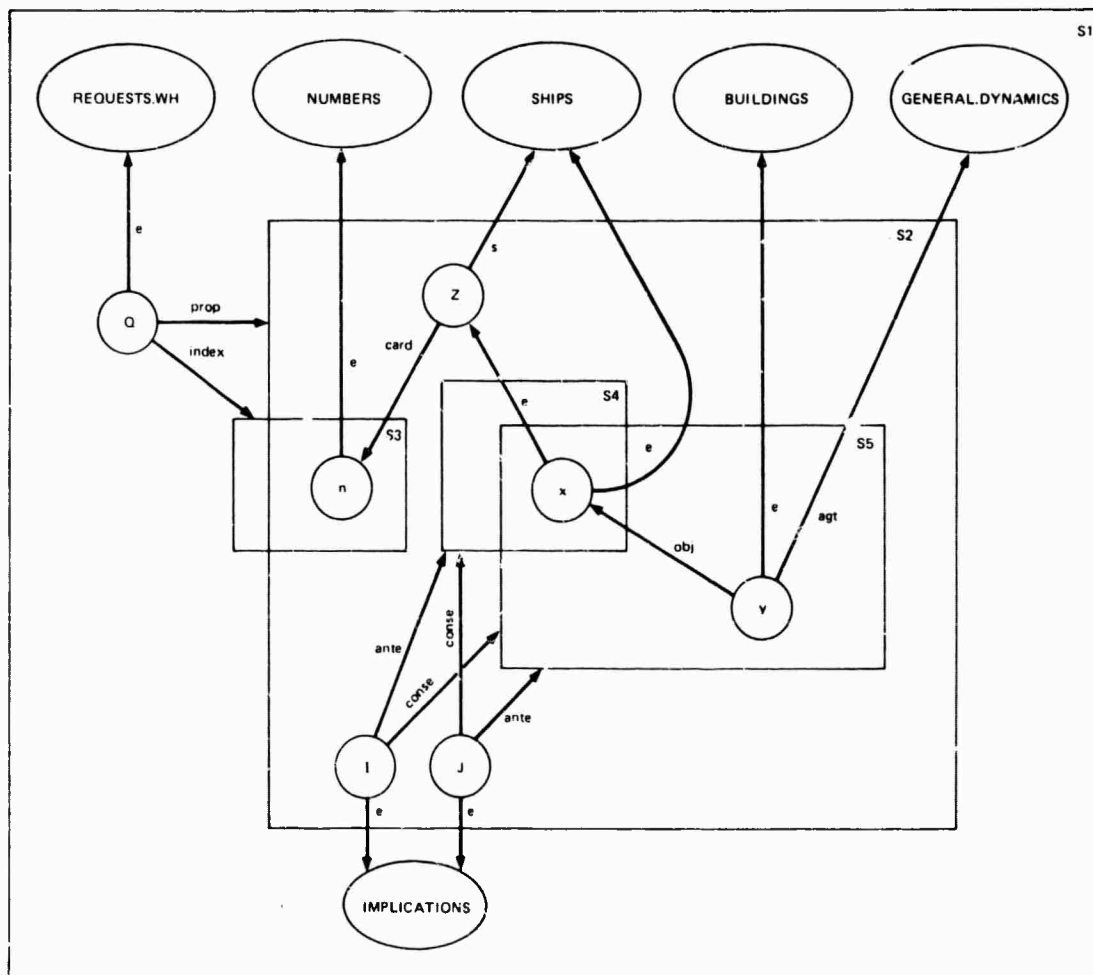


FIGURE V-36 HOW MANY SHIPS DID GENERAL.DYNAMICS BUILD?

F. AUGMENTATIONS

Since partitioned semantic networks are capable of encoding arbitrary logical statements, they have a high degree of completeness. Nevertheless, it may be more economical or convenient to encode certain types of information on property lists or in procedures. Therefore, the following features are offered as augmentations of the basic capabilities.

1. PROPERTY LISTS

Each of the various network entities (nodes, arcs, and spaces) has one or more property lists. Nodes and spaces have so-called "global" property lists that are like the property lists of atoms in LISP. In addition, arcs and nodes (including supernodes, i.e., spaces that have been given node-like features) have context-sensitive property lists resembling (and modeled after) the property lists of QLISP (Reboh and Sacerdoti, 1973). The context sensitivity of these property lists has been designed to parallel the visibility hierarchy of vistas described earlier. In particular, each item on one of these lists has a double key consisting of a property name and a space in a stratified configuration. All GETs and PUTs are made with respect to a vista. If a value V is stored under property P with respect to vista (S1 S2 ...), then the value is indexed by both P and the bottom space of the vista, S1. To GET the value of property P with respect to vista (S1 S2 ...), the system first checks to see if P has a value on space S1. If

it has, that value is taken. Otherwise, the other spaces in the vista are considered in order.

2. PROCEDURAL AUGMENTATION

There are a number of ways in which procedures may be linked with partitioned net structures to form procedurally augmented, partitioned semantic networks. Perhaps the most overt of these is the method used in the SRI speech understanding system. To see what this method is, consider the set SUMS, the set of all situations with three participants, addend1, addend2, and total, in which the total is the sum of the addends. Certain elements of this set might be represented explicitly in the network, but it would be impossible to explicitly encode the entire set. On the other hand, the INTERLISP function PLUS comes very close to modeling all the instances of practical interest. What is needed, then, is some way to use PLUS to create instances of SUMS on the fly.

Moving in this direction, let APPLICATIONS be the set of all situations in which an INTERLISP function is applied to an ordered set of arguments to produce a result. Then the general rule encoded in Figure V-37 shows the interrelation of PLUS and SUMS. Namely, for every x, y, and z, if z is the result of applying PLUS to x and y, then z is the total, x is the addend1, and y is the addend2 of a SUMS situation j. (A more cautious formulation would restrict x and y to be numbers.)

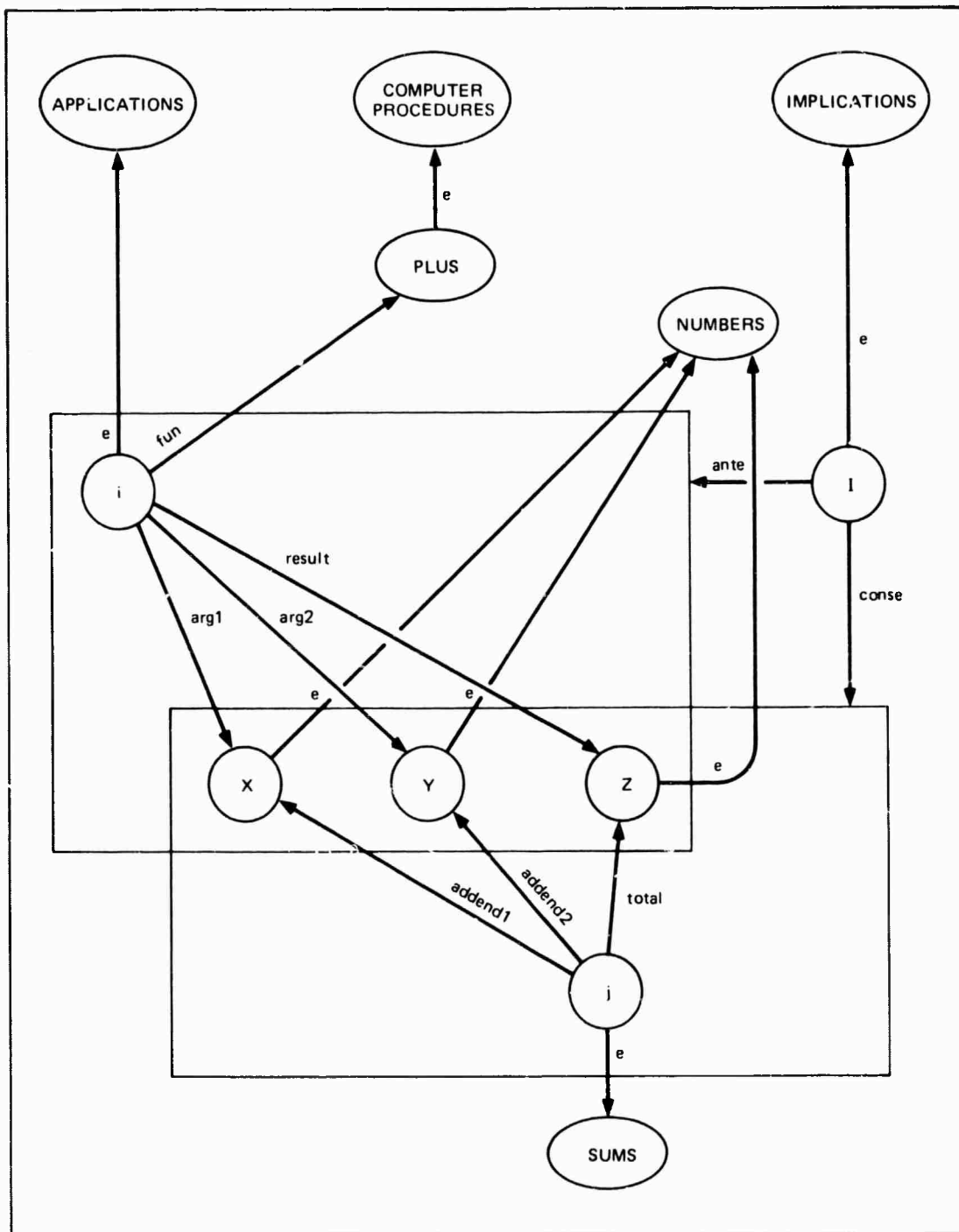


FIGURE V-37 RELATING SUMS SITUATIONS TO FUNCTION PLUS

Now the general rule of Figure V-37 merely transforms the problem of finding a SUMS situation into the problem of finding an APPLICATIONS situation. Were there no special mechanism for finding APPLICATIONS, one unsolvable problem would simply be replaced by another. But the algorithms that perform logical deduction in networks (described in Chapter XII) have special knowledge of the set APPLICATIONS. In particular, they know how to apply the function to the arguments to produce both a result and a new member of the APPLICATIONS set.

The bulk of specific information about the physical attributes of ships that is maintained by the SRI speech understanding system is kept on files and retrieved upon demand by file access functions.* These access functions are typical of a large class of functions that take an A-list as their only argument and return an A-list (or, more generally, a possibly empty list of A-lists) as their result. For example, the retrieval function SHIPDATA might be applied to the argument

(NAME Henry.L.Stimson OWNER ? BUILDER ?)

and return the resulting A-list

(OWNER The.U.S BUILDER General.Dynamics).

This application would then correspond to that depicted in Figure V-38. In this figure, the A-lists are encoded by nodes, with attribute/value pairs being encoded by outgoing arcs. Each arc's label carries the attribute and the arc's to-node carries the value.

* The file access system and its link to the network were written by Jonathan Slocum.

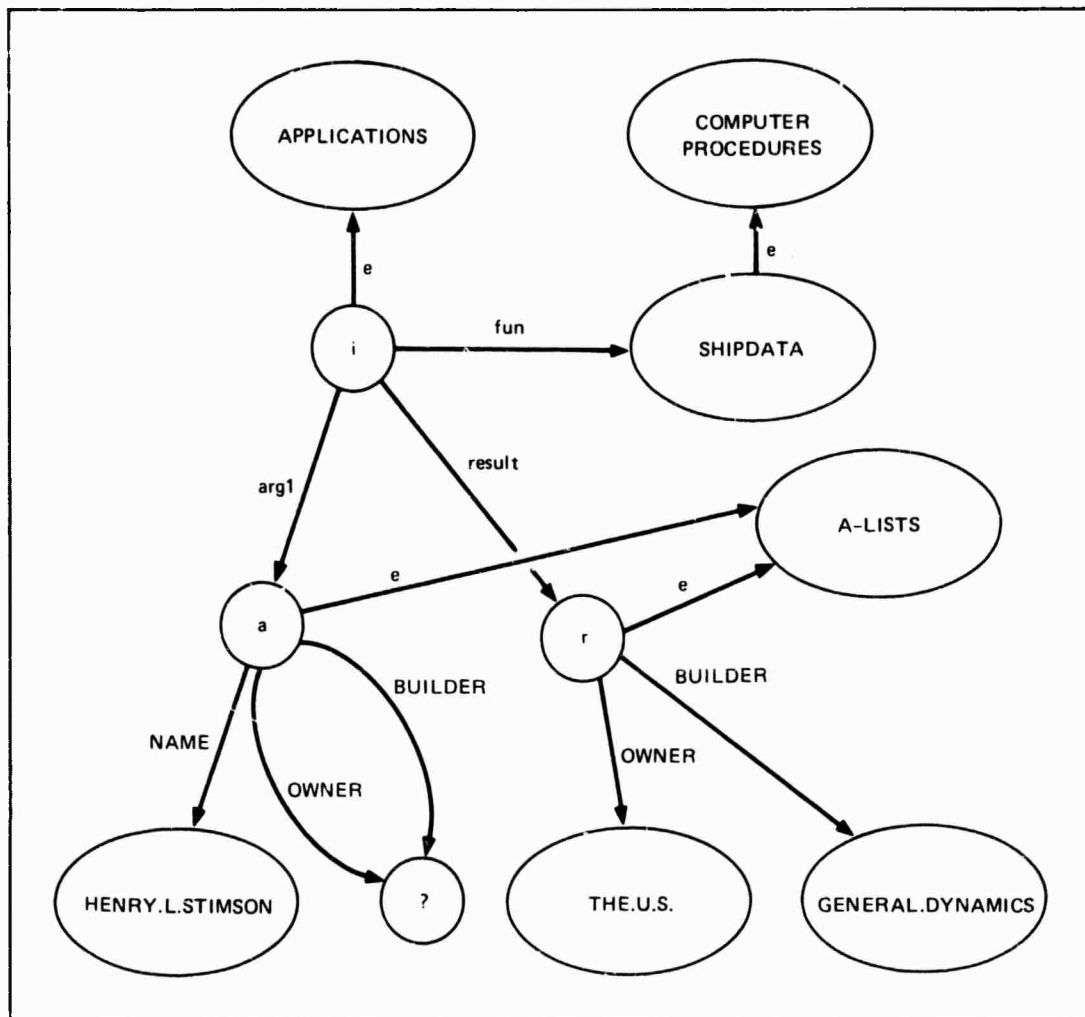


FIGURE V-38 AN APPLICATION OF SHIPDATA

Function SHIPDATA turns out to be very flexible. In fact, it is capable of taking just about any A-list of the form

(C1 V1 C2 V2 ... Cn Vn)

where the C_i are the names of columns on the file records and the V_i are either values to be matched or question marks. The function returns an A-list with entries for those i whose V_i were originally question marks.

Since the flexibility of SHIPDATA makes possible a large number of different calling configurations, the following expediency (i.e., hack) was used to compress the number of necessary general rules. A new situation set called KEYED-APPLICATIONS was defined. Each member of this set was associated with a function (usually SHIPDATA) and a number of cases. Furthermore, and this is the expediency, on the property list of each member there was placed a set of keys, where each key is a list of case names. The deduction algorithms were especially programmed to know about members of KEYED-APPLICATIONS. In particular, they were given the knowledge that the function could be called if all the cases listed in any key had values. The A-list to be used in this call consisted of all case/value pairs for which the values were known, supplemented by case/question-mark pairs for cases with unknown values.

Thus, a general rule involving a KEYED-APPLICATIONS situation with n keys could take the place of n general rules. For example, the general rule of Figure V-39 carries the force of three general rules, the first of which is shown in Figure V-40.

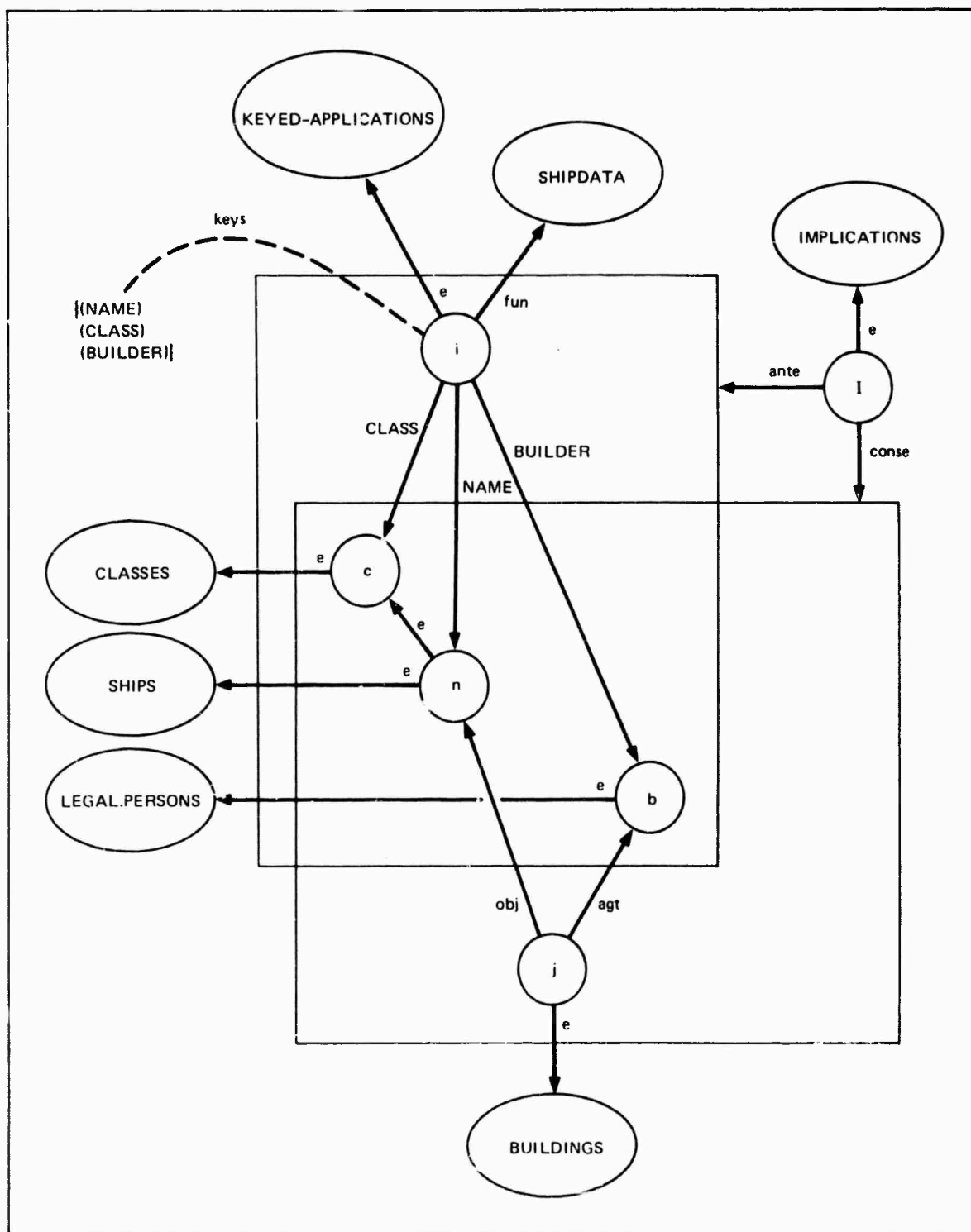


FIGURE V-39 A KEYED-APPLICATIONS SITUATION

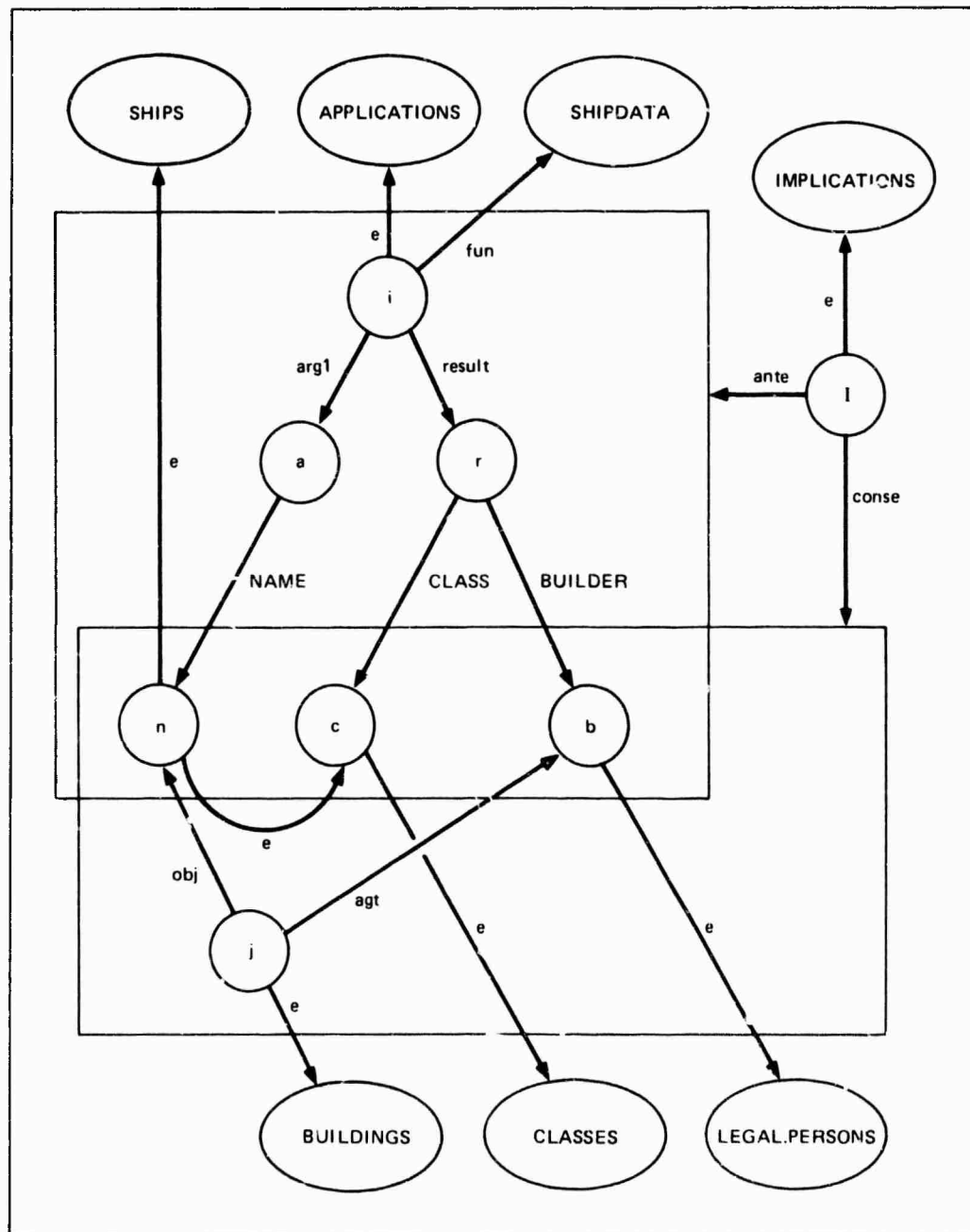


FIGURE V-40 THEOREM IMPLIED BY FIRST KEY

G. SUPPORTS FOR DIVERSE TASKS

The discussion of partitioned semantic networks presented thus far has emphasized mechanisms for encoding logical statements in nets, but the representation scheme supports other tasks as well.

1. FOCUS

One of these tasks is that of establishing local contexts for discourse analysis. Developed by Barbara Deutsch and described more fully in Chapter IX, the basic concept is to introduce a second partitioning of the network that complements the partitioning used in the encoding of logical connectives. Spaces in this second partitioning, called "focus spaces," are used to group together objects that have either been mentioned recently in the dialog or that are closely related to objects that have been mentioned.

By creating vistas of focus spaces, it becomes possible to define various levels of focus. Search algorithms may then begin with the most local information and proceed stepwise into larger and larger contexts.

2. SCRATCH SPACES

In the process of solving problems posed in the network formalism, it is often necessary to set up subproblems, consider alternative assumptions, and derive intermediate facts. (Such

intermediate facts typically consist of instantiations of quantified statements.) Although the network structures used to encode such intermediate information are of central interest during the problem solving process, such structures are of little value afterwards. Rather than clutter the network with intermediate results, the hypotheses and bits of derived information are placed in scratch spaces that serve as temporary extensions to the central model. After completing a problem solving activity, the "bottom line" may be moved to the central net and the extension forgotten. Details concerning the use of extension spaces are contained in Chapter XII.

3. RELATING SYNTAX TO SEMANTICS

As Figure V-8 shows, partitioning may be used to show the relationship between the syntax of an input and the input's translation in the net. This ability, as described in Chapter VII, is of central importance in the translation of quantified expressions and in allowing multiple parse-time hypotheses concerning the interpretation of an input to share network translations of subphrases. It is also crucial to the approach to processing elliptical expressions that is described in Chapter X.

H. LINEARIZED NET NOTATION

To communicate network structures to the computer, a linearized net notation, called the "LN2" language, has been devised as an extension of INTERLISP. The syntax of LN2 was inspired by and bears some resemblance to the syntax of KRL, the knowledge representation language of Bobrow and Winograd (1976).

To give an indication of the flavor of this language, an LN2 statement describing the network of Figure V-41 is presented in Figure V-42. Although this example illustrates only a fraction of the features of LN2, the central capabilities are covered.

The total statement is a call to function !SPACE of the form (!SPACE name e1 e2 ... en). Its first argument is a name to be given to a newly created space. All subsequent arguments are expressions to be executed in the context of the new space.

The first such expression is "[UNIVERSAL]", which read macros expand into "(!NODE UNIVERSAL)." In general, calls to !NODE are of the form (!NODE optional-name e1 e2 ... en). The function creates a new node on the current space, assigns it the optional-name (if any) and then evaluates the various expressions ei. Thus, [UNIVERSAL] just creates a node named UNIVERSAL.

"[SITUATIONS (ARE UNIVERSAL)]" creates a node named SITUATIONS and then executes the expression "(ARE UNIVERSAL)," which creates a ds arc

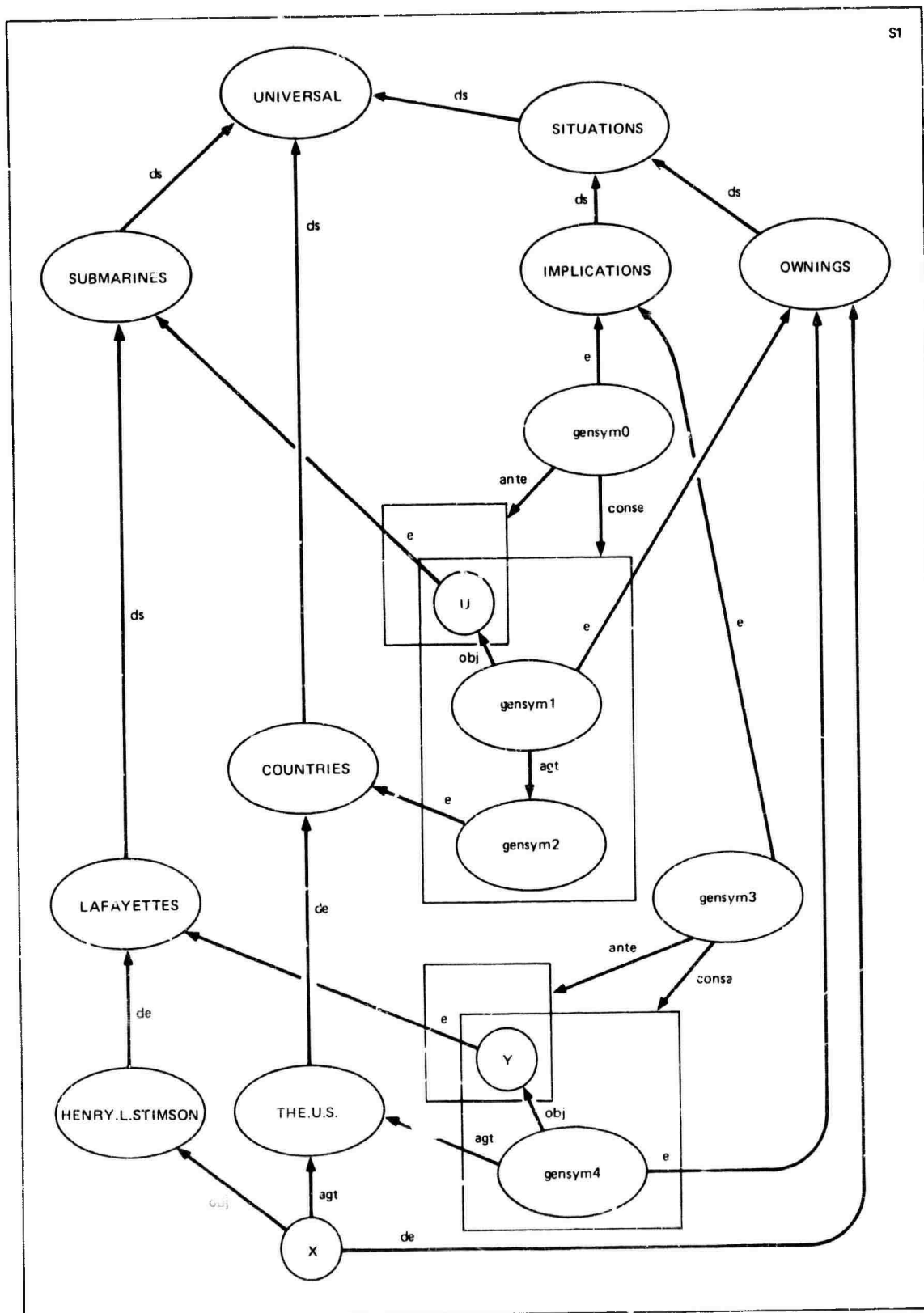


FIGURE V-41 NETWORK CREATED BY LN2

```

(!SPACE S1
  [UNIVERSAL]
  [SITUATIONS (ARE UNIVERSAL)]
  [IMPLICATIONS (ARE SITUATIONS)]
  [OWNINGS (ARE SITUATIONS)]
  [SUBMARINES (ARE UNIVERSAL)]
  [LAFAYETTES (ARE SUBMARINES)]
  [Henry.L.Stimson (A LAFAYETTE)]
  [COUNTRIES (ARE UNIVERSAL)
    (SINGULAR COUNTRY)]
  [The.U.S. (A COUNTRY)]
  [x (AN OWNING)
    {agt The.U.S.}
    {obj Henry.L.Stimson}]
  (TURN.OFF.D)
  (IMPLICATION
    ([u (A SUBMARINE)])
    ([ (AN OWNING)
      {obj u}
      {agt (A COUNTRY)}}]))
  (DEC.SIT
    owns (OWNER OWNEE)
    [(AN OWNING)
      {agt OWNER}
      {obj OWNEE}])
  (IMPLICATION
    ([y (A LAFAYETTE)])
    (<owns The.U.S. y>)) )

```

Figure V-42. AN LN2 STATEMENT

from the current node to 'UNIVERSAL'. The next four !NODE expressions are similar.

"[Henry.L.Stimson (A LAFAYETTE)]" causes a node to be created named Henry.L.Stimson. Function A produces a de arc from this node to the node whose name is formed by adding "S" or "ES" to the argument of A. Hence, the de arc from 'Henry.L.Stimson' to 'LAFAYETTE'.

Since COUNTRY has an irregular plural, the expression creating node 'COUNTRIES' has a call to function SINGULAR to note this fact. SINGULAR does the necessary bookkeeping so that the call to A in "[The.U.S. (A COUNTRY)]" works properly. (Note: words and spellings used by LN2 have nothing to do with the lexicon of the SRI speech understanding system.)

"[x (AN OWNING) {agt The.U.S.} {obj Henry.L.Stimson}]" creates node 'x', encodes x as a distinct element of OWNINGS, and then creates an agt arc to 'The.U.S.' and an obj arc to 'Henry.L.Stimson'.

The expression "(TURN.OFF.D)" changes the operation of functions A and ARE so that de and ds arcs are replaced subsequently by e and s arcs.

Function IMPLICATION takes two arguments: a list of expressions for creating structures inside an implication ante space and a similar list for the conse space. IMPLICATION builds a new element of IMPLICATIONS with appropriate new spaces and then executes the lists of expressions. New structures created or referred to by both ante and conse are placed in the overlap.

In the first IMPLICATION of the example, the ante space expressions (there is only one) cause a node labeled "u" to be created with an e arc to SUBMARINES. The sole conse space expression calls for a node to be created and assigned a gensym name. The node represents an element of OWNINGS. The obj of this element is u. The agt is to be encoded by a newly created, gensym named node with an e arc to COUNTRIES.

Function DEC.SIT (= declare situation) creates no structure itself but defines shorthands for subsequent use. The example shown defines "owns" situations in terms of the local variables (formal parameters) OWNER and OWNEE. The remaining arguments to DEC.SIT are expressions to be evaluated when an owns situation is invoked. That is, this call to DEC.SIT defines a type of subroutine for creating network encodings of owns situations.

An invocation of owns occurs in the conse of the last IMPLICATION. The delimiters "<" and ">" indicate that a situation is to be instantiated. The first argument within the delimiters is the situation name (which must have been previously declared in a DEC.SIT) and the other arguments are actual parameters for the situation subroutine.

I. APPLYING THE REPRESENTATION

This chapter has outlined a method for encoding a variety of types of information in procedurally augmented, partitioned semantic networks. However, this is only the first half of a two-part story. As important as the ability to represent information is the ability to apply the information to the performance of tasks. This other half of the story, in particular, is the subject of the chapters on semantic translation (Chapter VII) and on deduction (Chapter XII).

VI THE MODEL OF THE DOMAIN

Prepared by Gary G. Hendrix

Using the partitioned semantic network formalisms described in Chapter V, a model was constructed for the data base-oriented domain the SRI speech understanding system. This model consists almost entirely of information about ships in the U.S., Soviet, and British fleets. Seventy-six classes of ships are included, covering 740 individual ships, over 200 of which are known by name. Such characteristics as the owner, builder, length, beam, draft, displacement, number in crew, speeds (surface and submerged), class, and type are available for each ship. In all, more than 30 relationships about ships are considered.

At the top level, this domain model is encoded as a large conjunction of individual facts and general rules. A small portion of the space, called the "KNOWLEDGE" space, that encodes this top-level conjunction is shown in Figure VI-1. In particular, this figure shows the top levels of the model's hierarchical taxonomy. The model divides the UNIVERSAL set into seven major disjoint subsets, which will be discussed below.

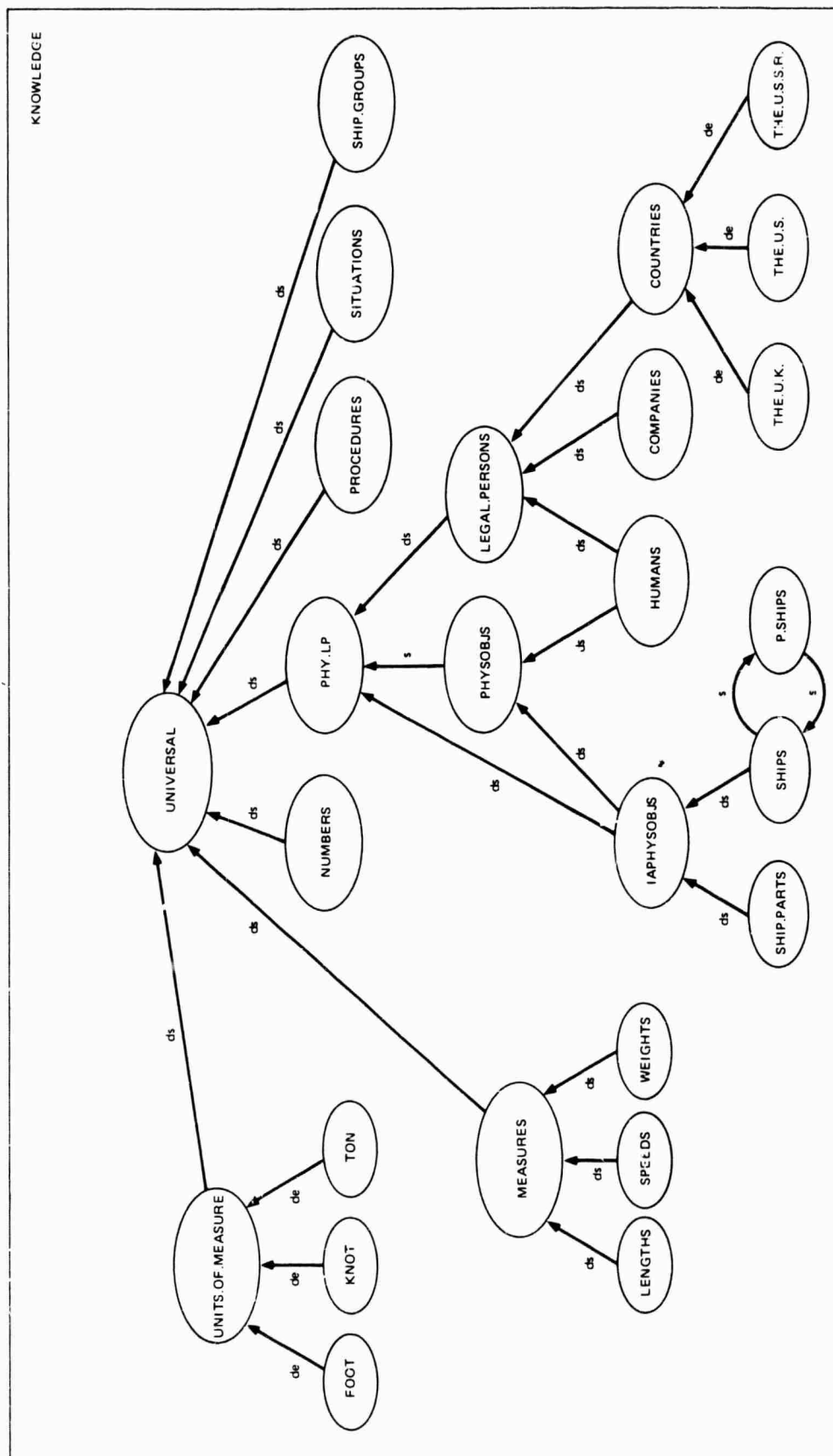


FIGURE VI-1 TOP OF DOMAIN MODEL

Several of the ship properties considered in the task domain are physical characteristics, such as length and weight, that are quantitative in nature. To deal with such dimensioned quantities, the sets UNITS.OF.MEASURE, NUMBERS, and MEASURES are included in the model. As Figure VI-2 shows, the delineation of MEASURES indicates a close relationship between these three sets. In particular, any D.MEASURE, an element of MEASURES, will be associated with two component parts: a num (= number) n taken from the set NUMBERS, and a unit u taken from UNITS.OF.MEASURE.

Certain subsets of MEASURES particularize the units. For example, the delineation of LENGTHS shows the unit to be restricted to FOOT. (A more general system would allow INCHES, METERS, and the like.) Since the number of numbers and measures is infinite, only those numbers and measures that are needed to encode other relationships are included in the network. Routines that do translation, numeric computations, and retrieval from files can produce new number and measure nodes on demand.

Returning to the hierarchy shown in Figure VI-1, consider the set PHY.LP, whose principal reason for inclusion in the model is to show (through s and ds arcs) the relationships between PHYSOBSJS (the set of physical objects), IAPHYSOBSJS (the set of inanimate physical objects), and LEGAL.PERSONS (the set of HUMANS, COMPANIES, COUNTRIES, and the like). IAPHYSOBSJS and LEGAL.PERSONS, being disjoint subsets of PHY.LP, have no elements in common. PHYSOBSJS is a subset of PHY.LP that includes all of IAPHYSOBSJS and that shares HUMANS in common with

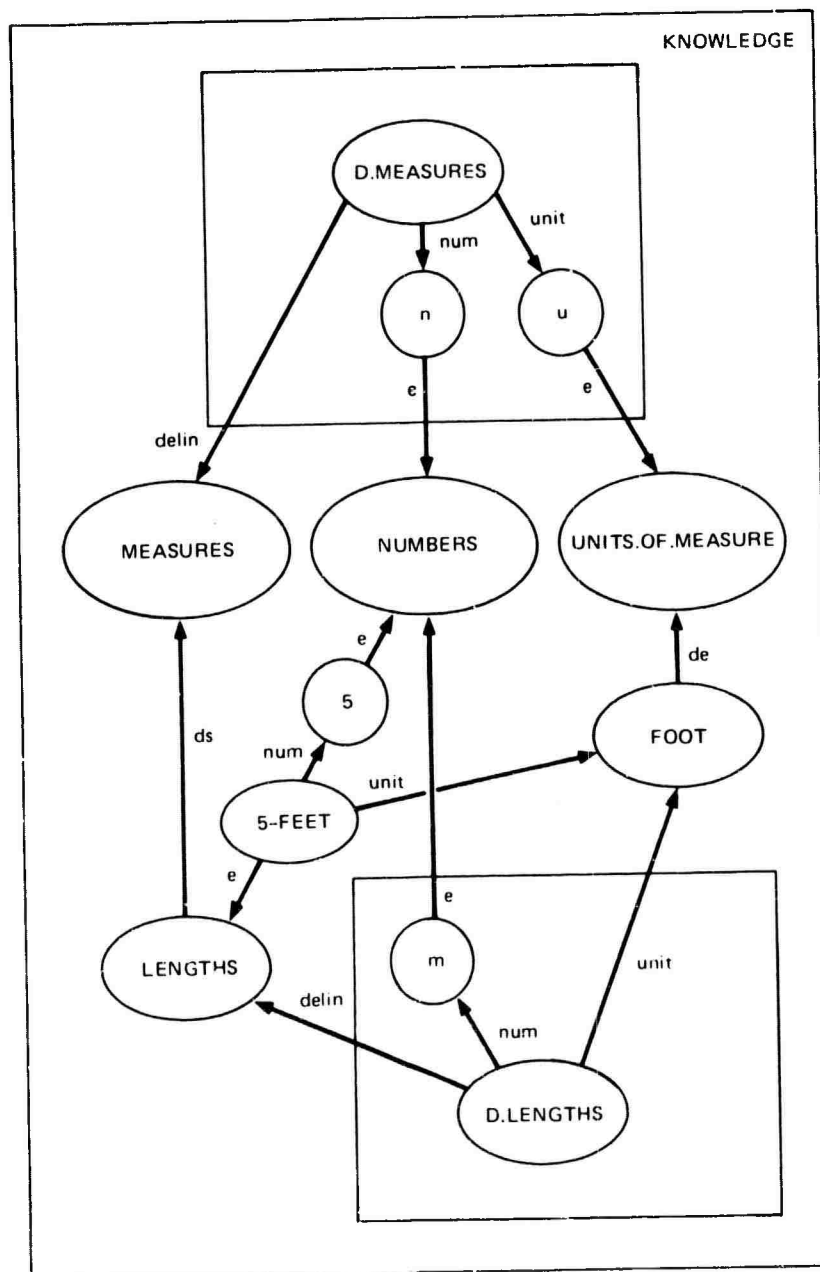


FIGURE VI-2 DELINEATIONS OF MEASURES AND SPEEDS

LEGAL.PERSONS. IAPHYSOBSJS includes SHIP.PARTS and SHIPS. SHIP.PARTS, not shown in Figure VI-1, includes POWER.PLANTS, REACTORS, TURBINES, TORPEDO.LAUNCHERS, and the like. Also not shown are 21 elements of COMPANIES.

Since the focus of the task domain is ships, the set SHIPS is of central importance in the model, and most of the nodes of the network are used in encoding its subsets or elements. Of equal importance with SHIPS is SHIP.GROUPS, the power set (set of all subsets) of SHIPS. A small but illustrative fraction of the overlapping taxonomies of these two sets is shown in Figure VI-3.

The set SHIP.GROUPS is divided into three major disjoint subsets: CLASSES, TYPES, and MACRO.GROUPS. The set of classes is composed of members that are sets of ships that are all basically identical. In particular, for this data base all the members of a given class are considered to have been made by the same manufacturer to the same specifications. TYPES are more general sets, grouping together ships that have similar (but not identical) characteristics and purposes. For example, the type SSBN is a set containing all ballistic missile submarines that are nuclear powered. This type includes all members of the ETHAN.ALLEN, GEORGE.WASHINGTON, LAFAYETTE, and RESOLUTION classes. While ships of the same type are by no means identical (a Lafayette is 65 feet longer than a Resolution), their military characteristics are closely related. Nodes representing the various types are labeled with the abbreviations commonly used by the Navy. Even more general sets

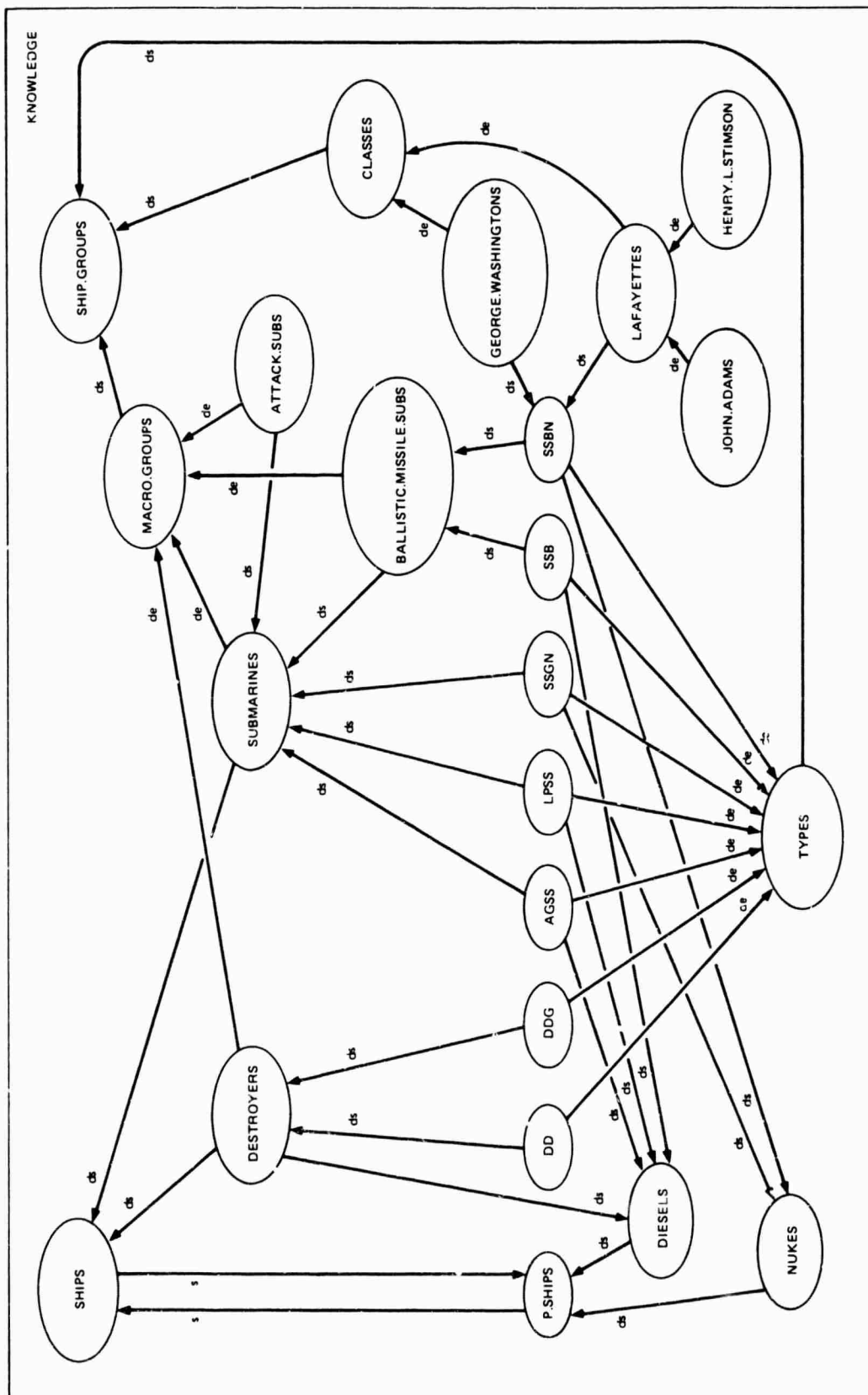


FIGURE VI-3 THE OVERLAPPING TAXONOMIES OF SHIPS AND SHIP GROUPS

than TYPES are MACRO.GROUPS. These include broad categories such as DESTROYERS and SUBMARINES.

The model includes 67 classes and 23 types, covering over 700 individual ships. Of these, 201 are explicitly recorded in the network by nodes with de arcs into one of the classes. For example, 'Henry.L.Stimson' has a de arc to 'LAFAYETTES', which indicates that the Henry.L.Stimson is a member of the Lafayette class. Tracing the membership of the Henry.L.Stimson through the more general sets provides the following information: it is of the SSBN type; it is a ballistic missile sub; it is a submarine; and it is a ship. Taking the second ds arc from 'SSBN', the Henry.L.Stimson is also shown to be a nuke (nuclear powered).

Although the types and macro.groups form disjoint subsets, the network encodes no explicit explanation of criteria used to define (and hence to distinguish) these sets. For example, there is no indication that the set of aircraft carriers is exactly that set of ships providing runways for airplanes. Although the network formalism is fully capable of encoding such information, the kinds of interactions with the data base in the current domain did not require it, so it is not included in the current model.

As indicated by the two s arcs that together connect 'SHIPS' and 'P.SHIPS' in both directions, these nodes of Figure VI-1 and Figure VI-3 represent the same set. The inclusion of two nodes is to allow ds arcs

to encode two divisions of the ships into disjoint subsets. The first group of disjoint subsets (those with ds arcs into 'SHIPS') includes DESTROYERS, SUBMARINES, FRIGATES, CARRIERS, and CRUISERS. This division is based on ship function. The second group (those with ds arcs into 'P.SHIPS') includes DIESELS and NUKES. This division is based on the kind of fuel that supplies power to the ships.

Let us return to Figure VI-1 for a final look; the last two major subsets of UNIVERSAL are COMPUTER.PROCEDURES and SITUATIONS. COMPUTER.PROCEDURES includes those computer codes that are used in connection with the APPLICATIONS feature of network procedural augmentation (as described in Chapter V, Section F.2). SITUATIONS is the set of all situations (relationships) existing in the ship domain. Almost forty subsets of SITUATIONS are included in the model. Some of these are used for internal purposes, including IMPLICATIONS, DISJUNCTIONS, NEGATIONS, REQUESTS.YN, REQUESTS.WH, APPLICATIONS, and KEYED-APPLICATIONS. The remainder of the sets model categories of situations that may be talked about in the language accepted by the SRI speech understanding system. These include such categories as OWNINGS, BUILDINGS, HAVE.PART, GREATER.THAN, HAVE.LENGTH, HAVE.SURFACE, DISPLACEMENT, and PRED.TRAINING. All sets in the last group have delineations to aid the translation process.

The several situation sets having names of the form "HAVE.<dimension>" are used to encode the situations of a physical object having a certain dimension to its character that is associated

with a quantitative measure. For example, Figure VI-4 shows the delineation of HAVE.BEAM. This delineation indicates that members of HAVE.BEAM relate a physical object p to a length q.

The several situation sets having names of the form "PRED.<property>" are used to encode situations in which some object has the property <property>. For example, members of PRED.TRAINING indicate that the participant filling their "pobj" case is used in training. (The system currently has no detailed model of what training is.)

All of the situations in this task domain turned out to be either binary or unary. A limitation to such simple situations was neither planned nor desired; it just happened. The networks themselves are capable of handling situations with arbitrary numbers of participants and actually become more efficient as the number of participants increases.

The encoding of information about the participation of particular ships in the various categories of situations is handled almost exclusively by universally quantified statements. Some of these statements make no appeal to sources of knowledge outside the network. For example, the statement of Figure VI-5 makes no appeal to external knowledge in indicating that every ship of type CVT is used in training. But most of the quantified statements used in the model rely on the KEYED-APPLICATIONS feature to access information in a relational data base. For example, the general rule of Figure VI-6 indicates

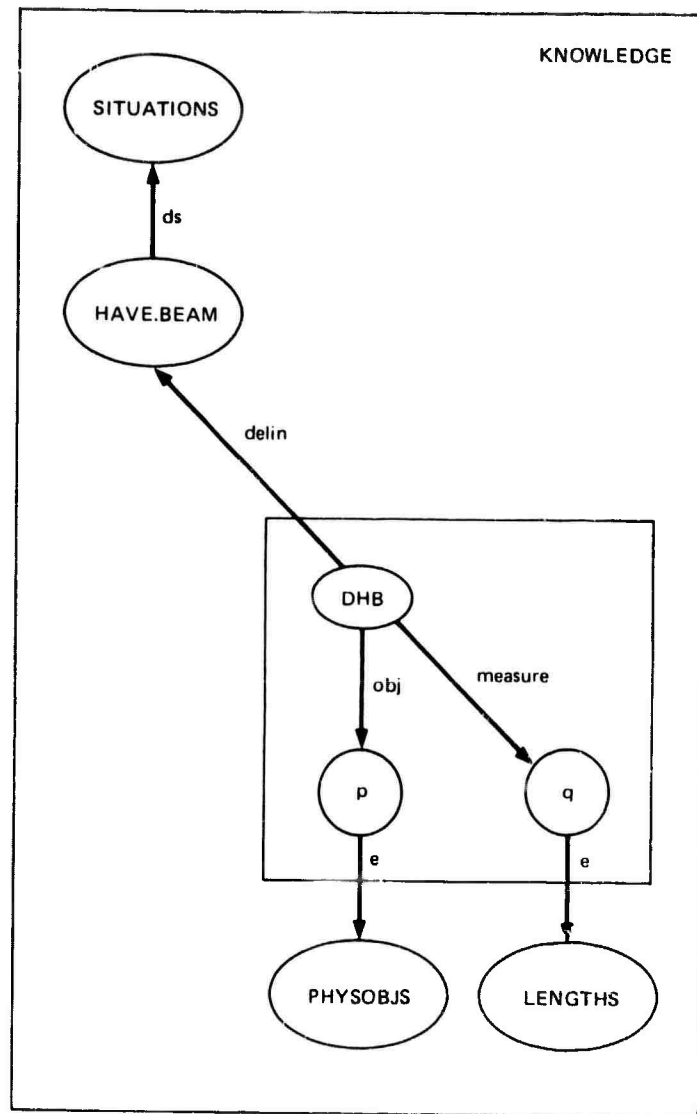


FIGURE VI-4 THE DELINEATION OF HAVE.BEAM

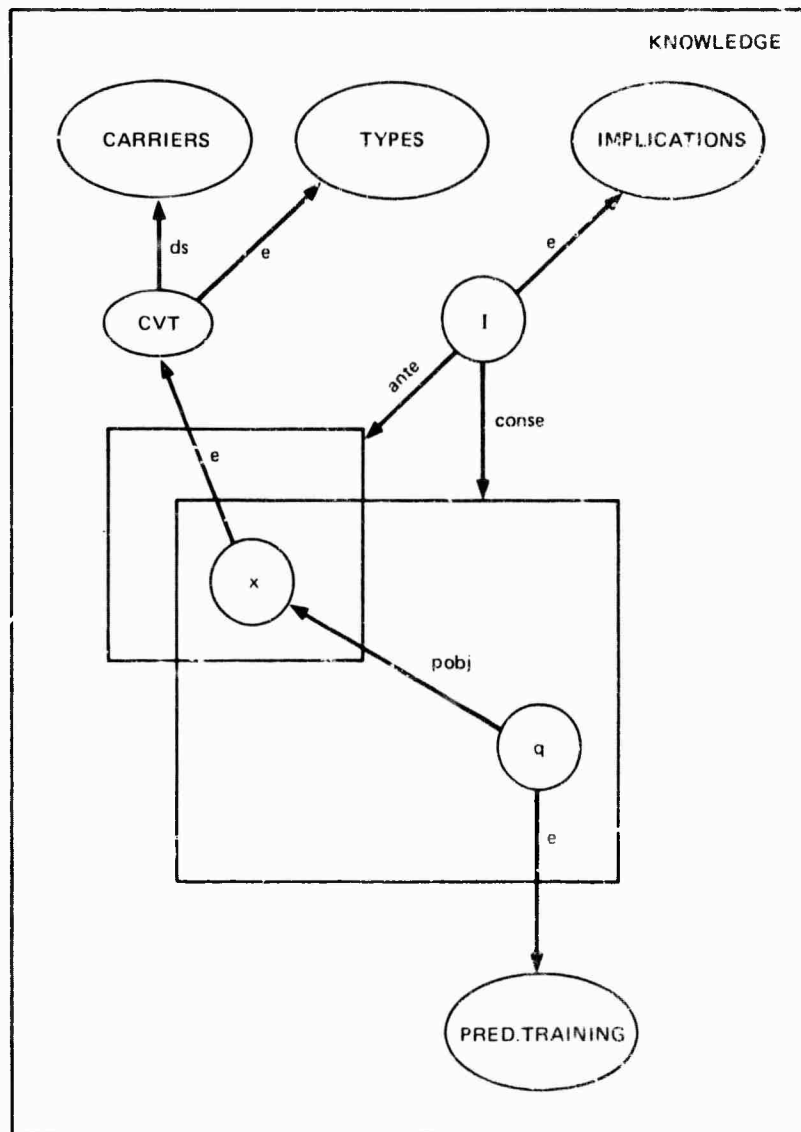


FIGURE VI-5 ALL CVTs ARE TRAINING SHIPS

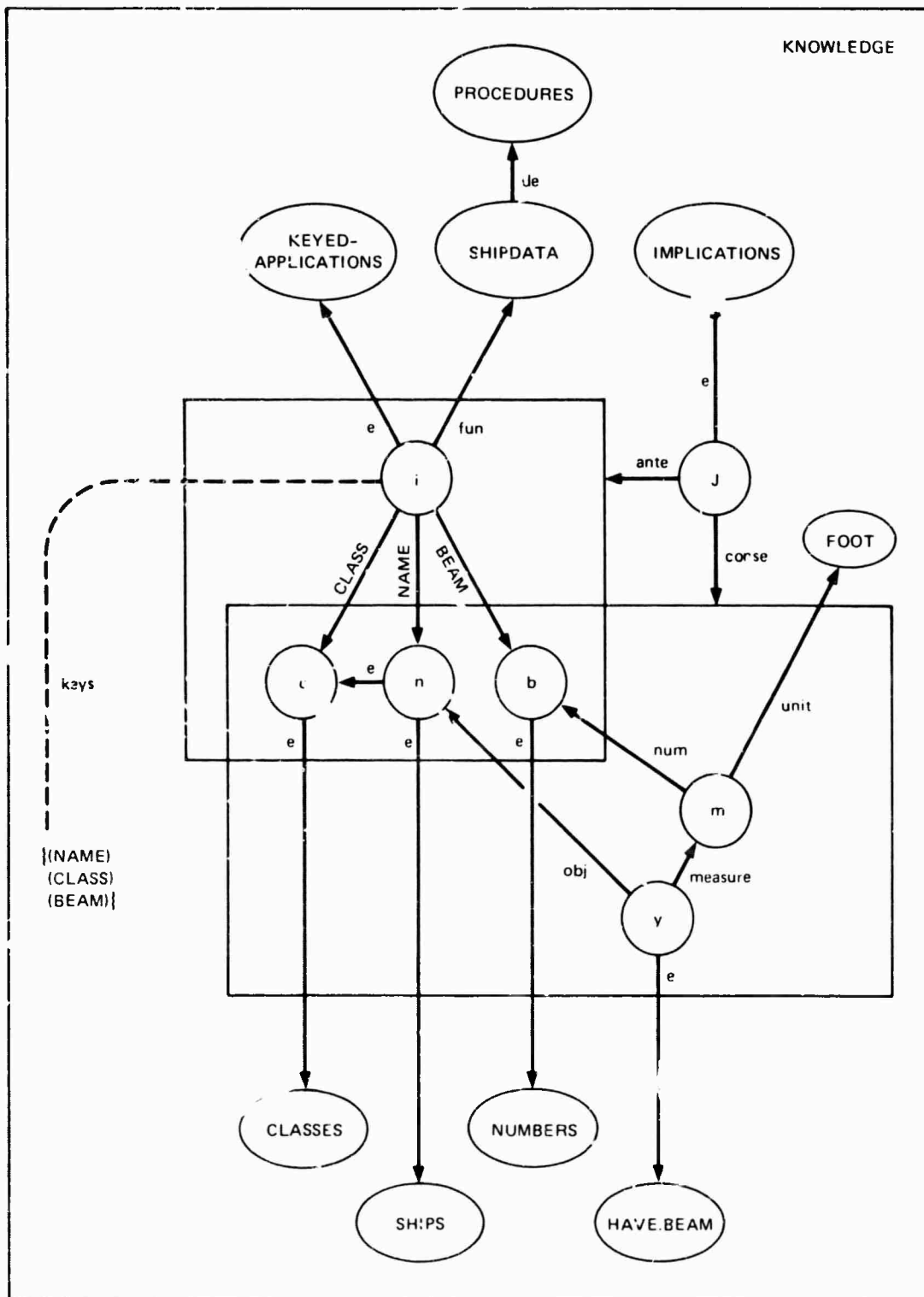


FIGURE VI-6 LINKING HAVE.BEAMS SITUATIONS TO RELATIONAL FILES

that a number giving the beam of a ship in feet may be found by calling function SHIPDATA with the argument

(NAME x BEAM ? CLASS ?),

where x is the ship whose beam is to be found. (The keys indicate two other possible calls: given a class, beams and individual ships may be retrieved; given a beam, classes and individuals in the class may be retrieved.) Not all of the KEYED-APPLICATIONS use function SHIPDATA. For example, the set GREATER.THAN is linked to an arithmetic procedure.

The construction of a model for the navy ships domain that has some degree of completeness was undertaken primarily to provide a foundation for semantic processing in the speech understanding system. However, this detailed development of a particular model provided a test of the representation scheme described in Chapter V and suggested some useful extensions. In particular, the notion of KEYED-APPLICATIONS and the use of de and ds arcs were directly motivated by the experience of building a model for the ship data. In earlier stages of the project, the representational scheme also was used in building a fragmentary model of the steps involved in assembling and disassembling an air compressor.

VII SEMANTIC ASPECTS OF TRANSLATION

Prepared by Gary G. Hendrix

CONTENTS:

- A. Introduction
 - 1. Integration of Filtering and Structure Building
 - 2. Timely Semantic Filtering
 - 3. A Two-phase System
 - 4. Cooperation with discourse
- B. Phase I: Semantic Composition
 - 1. An Introductory Example
 - 2. Technical Comments on the Example
 - a. Sharing Network Structures
 - b. Syntactic Order
 - c. Conceptual Spaces
 - d. Communicating with the System Executive
 - 3. Interacting with Discourse: Determined Noun Phrases
 - 4. Other Aspects of the SCRs
 - a. Multiple Node Lexical Entries
 - b. Equiv Arcs
 - 5. More on Delineations
 - 6. Semantic Composition Rule Summary
- C. Phase II: Quantification
 - 1. Overview
 - 2. The Quantifiers
 - 3. Space Ordering
- D. The Use of Case Information

A. INTRODUCTION

The subject of this chapter is the utilization of semantic knowledge in the process of understanding spoken inputs as practiced in the SRI speech understanding system. Basically, there are three functions that a semantic component may perform during the understanding process. First, it may filter out phrase combinations that, although syntactically and acoustically acceptable, do not meet semantic criteria for meaningful unification. Second, for combinations that are acceptable, the semantic component may build deep, internal structures representing the meaning of the input (or portions of the input) in the context of a particular task domain. Third, by considering the meaning of a phrase that constitutes a fragment of the utterance, the semantic component may make predictions concerning what words or syntactic constructions are likely to occur in other parts of the utterance. In the current implementation of the speech understanding system, the first two of these functions, filtering and structure building, are performed in a single module; prediction of likely words (but not of syntactic constructions) is carried out in a separate procedure. In this chapter, the major emphasis will be on the first two; prediction will be treated briefly at the end.

To understand the details of semantic filtering and structure building, it will be helpful to consider first some of the higher-level design features of the system.

1. INTEGRATION OF FILTERING AND STRUCTURE BUILDING

One of the system design features to note is that filtering and structure building are not handled as individual processes but are treated collectively. It would be convenient to have semantic filtering guide the parsing while saving the relatively expensive structure building task for a postparsing phase in which the syntactic analysis of an input would be known in total and the building of structures for spurious phrases could be omitted. However, it turns out that filtering (by both semantics and discourse) is dependent upon the structures assigned to subphrases of the input. Therefore, filtering and structure building are combined. When a phrase combination is proposed to the semantic system, the system attempts to build up a structure encoding the meaning of the new phrase. If any of various checks and restrictions in the structure-building process recognize an anomalous condition, the structure building fails, and this failure, acting as a filter, serves to reject the phrase combination.

2. TIMELY SEMANTIC FILTERING

Most text-based understanding systems (e.g., Woods et al., 1972) perform a complete syntactic analysis of an input before taking any but the most superficial semantic considerations (e.g., number agreement) into account. This approach is quite reasonable, since, for processing text, semantic analysis tends to be far more expensive than reading words from the input buffer or manipulating the grammar.

However, when dealing with the added costs and uncertainties of acoustic input, the early use of semantic filtering (and, consequently, structure building) to prune misheard words and false paths through the grammar becomes more attractive. Therefore, in the SRI speech understanding system, the semantic component is given the opportunity to reject each new phrase when it is first proposed.

3. A TWO-PHASE SYSTEM

An additional design feature, which has had a great influence on the overall structure of the semantic system, is that the scoping of quantified variables is saved for a postparsing phase. There are two reasons for this postponement. First, the determination of scopes is extremely context sensitive, making it difficult (or impossible) to perform in a bottom-up fashion, one phrase at a time. Second, the information that scoping adds to the structures representing the semantic interpretations of phrases provides few (if any) new clues that are helpful in filtering. Thus, it is also more efficient to delay the quantification process.

4. COOPERATION WITH DISCOURSE

The semantic and discourse components of the speech understanding system are closely coordinated and should be studied as a pair. Both components build and evaluate networks that describe the system's interpretation of phrases in the input. Interpretations for

some phrases (e.g., pronouns) are built by discourse alone. Some types of phrases (e.g., indefinite noun phrases, verb phrases, prepositional phrases) have interpretations constructed by the semantic component alone. But some phrases (e.g., definitely determined noun phrases) are interpreted by a cooperative effort in which the semantic component builds an intentional description of the phrase's meaning, and discourse relates this intentional description to a particular object in the domain model. In forming an interpretation for a new composite phrase, the semantic module uses the interpretations for each of the phrase's constituent subphrases. These interpretations may have been produced either by semantics or discourse (or have come directly from the lexicon).

As another point of cooperation between discourse and semantics, after discourse expands an elliptical input into a sentence level interpretation, the semantic system is used to add quantification.

B. PHASE I: SEMANTIC COMPOSITION

As indicated above, the operations of the semantic component may be separated into two phases. The first of these phases, called the 'composition' phase, is the subject of this section. The second (or 'quantification') phase is discussed in Section C.

The task of the composition phase is to provide semantic filtering and (unquantified) structure building in support of the parsing process.

This task is performed by a battery of semantic composition routines (SCRs) that are tightly coordinated with the language definition of the system (see Chapter II). Whenever a language definition rule suggests the feasibility of combining a number of components of the input to produce a larger or more general phrase, one of these SCRs is invoked. Acting as a filter, the SCR may reject the combination on semantic grounds. If the combination is accepted, then the SCR builds a network structure representing the (unquantified) interpretation of the phrase. In building up such structures, the SCRs are in fact COMPOSING network paraphrases of the input phrases. Hence the name 'composition routine.'

Since different SCRs are associated with different rules of the language definition, each SCR constitutes a procedural encoding of the knowledge concerning the semantic import of the associated syntactic production(s). This procedural specification references and coordinates the declarative semantic knowledge in the system's lexicon and in the network-encoded domain model. Bringing these sources of semantic information to bear on a proposed phrase combination, an SCR creates an interpretation structure meeting a number of highly interdependent criteria. These include:

- * Creating an interpretation structure that accurately models the (unquantified) meaning of the phrase.
- * Reusing the network structures of components in building interpretations of the composite phrase. (This consideration, which is nontrivial in bidirectional networks, makes the building of the composite phrase less expensive.)

- * Building structures that allow multiple hypotheses concerning the proper incorporation of a given utterance component in larger phrases to be encoded simultaneously. (This makes possible the handling of ambiguous situations.)
- * Allowing competing users of a subphrase to share a single network structure representing the interpretation of that subphrase. (Recycling increases efficiency.)
- * Incorporating special quantification markers into the structure that are effectively invisible to other parts of the speech understanding system. (These markers are needed by the quantification phase but must be so encoded that they do not change the meaning of the unquantified structures.)
- * Indicating the association between each syntactic unit of the composite phrase and its contribution to the network interpretation structure. (This association is needed both by the quantification phase and by that portion of the discourse component that expands elliptical inputs.)

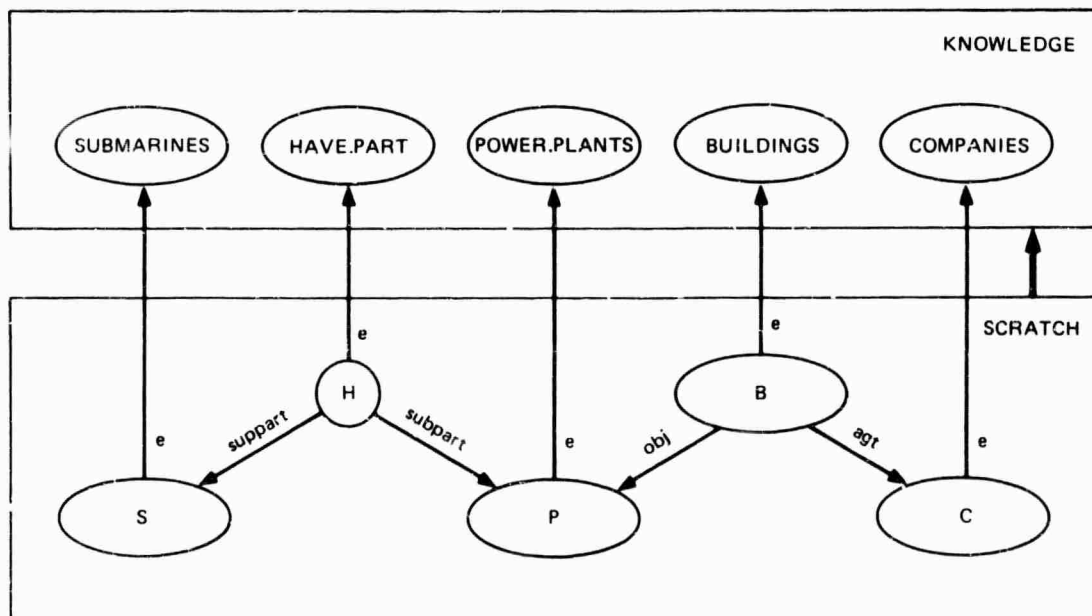
The ability to use the structures of subphrases in the building of composite structures and the complications of simultaneously maintaining multiple hypotheses make the interactions of the SCRs both more important and more interesting than the operation of any one SCR in isolation. Therefore, the operations of the SCRs will be presented by a series of examples in which many SCRs participate in the construction of an interpretation of a complete utterance.

1. AN INTRODUCTORY EXAMPLE

To introduce most of the important features of the SCRs while postponing side issues, consider, for the purposes of simplicity, the parsing of the following, rather unlikely, sentence:

"A power plant of a submarine was built by a company."

The ultimate result of the semantic interpretation process for this sentence is the network structure recorded in the SCRATCH space of Figure VII-1. Structures representing new inputs are constructed in a scratch space (or spaces) to prevent them from becoming confused with the system's model of the task domain, which is recorded on the KNOWLEDGE space. Since the SCRATCH space of the example is immediately below the KNOWLEDGE space in the viewing hierarchy (as shown by the heavy arrow), the view from the SCRATCH space includes the structures in the KNOWLEDGE space. In Figure VII-1, the scratch space is presented in its entirety, but only a fraction of the structures in the KNOWLEDGE space have been shown.



SA-3804-35R

FIGURE VII-1 PARSE TARGET STRUCTURE FOR "A-POWER-PLANT OF A-SUBMARINE WAS-BUILT BY A-COMPANY"

Since the system interprets new inputs by calling on previous knowledge, there are several links from the SCRATCH space into the KNOWLEDGE space. The interpretation of the network in the SCRATCH space is as follows: Node 'B' represents an element of the set BUILDINGS, the set of all building events. In the particular event B, an agt (agent) C is the builder of an obj (object) P. The agent C of the building event is an element of COMPANIES. The object built by C is P, an element of the set POWER.PLANTS. Node 'H' encodes the proposition that power plant P is the subpart in a HAVE.PART situation in which S, some member of the set of SUBMARINES, is the suppart (super part).

To suppress syntactic technicalities while concentrating on the semantic aspects of the construction of this interpretation structure from the original English input, assume the highly simplified language definition:

GRAMMAR

R1: S => NP VP
R2: NP => NP PREPP
R3: VP => VP PREPP
R4: PREPP => PREP NP

LEXICON

NP: a-power-plant,
a-submarine, a-company
VP: was-built
PREP: of, by

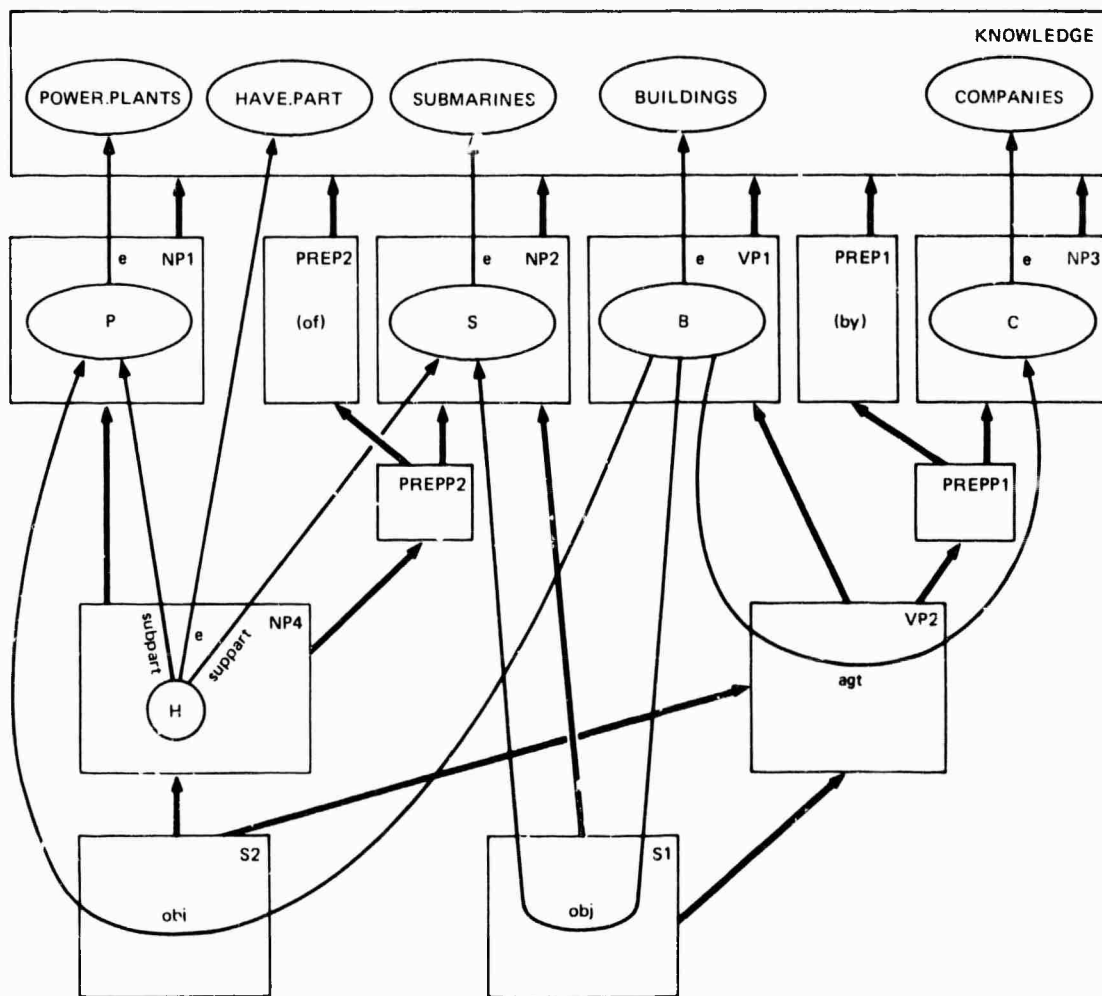
(NOTE: "a-power-plant" is not treated as an NP in the actual system. Rather, "power plant" is first combined with PREPP "of a submarine" and only afterward is "a" appended to produce the NP "a power plant of a submarine".)

In the translation process, spaces are created to represent the semantics of each grammatically defined constituent of the total utterance. These spaces are shown in Figure VII-2, with heavy arrows indicating the visibility hierarchy.

At the start of processing, space KNOWLEDGE contains knowledge about power plants, HAVE.PART situations, submarines, building events, and companies. Upon spotting the noun phrase "a-power-plant", an SCR is called to set up a structure representing the meaning of the phrase. In particular, the SCR creates a new space, NP1, below the KNOWLEDGE space in the viewing hierarchy. Within this space, a node 'P' is created with an e arc to 'POWER.PLANTS'. Thus, node 'P' represents some power-plant and the e arc makes its membership in POWER.PLANTS explicit. The new space NP1 separates the structures built to represent the phrase from structures that are in the KNOWLEDGE space. Similarly, new spaces PREP2, NP2, VP1, PREP1, and NP3 are set up to encode other utterance constituents that correspond to explicit lexical entries (terminals).

As language definition rules suggest the grouping of subphrases into larger units, SCRs are called to aid in the process. Using rule R4, PREP1 ("by") and NP3 ("a-company") are combined to form PREPP1 ("by a-company"). PREPP1 is allocated its own space, but no new structures are created within it.

When syntactic considerations suggest combining VP1 ("was-built") with PREPP1, the appropriate SCR is called. Consulting a



SA-3804-19R1

FIGURE VII-2 MULTIPLE SCRATCH SPACES FOR "A-POWER-PLANT OF A-SUBMARINE WAS-BUILT BY A-COMPANY"

surface-to-deep-case map associated with the lexical entry for the verb "build",* the SCR determines that a "by" PREPP following the verb often signals the deep agt case in a passive construction. Operating under this hypothesis, the SCR checks the voice of VP1. Passing this test, the SCR next checks the semantic feasibility of the NP of PREPP1 serving as the agt in a BUILDINGS event. To make this check, the SCR consults the delineation of BUILDINGS, which indicates that any agt of a BUILDINGS situation must be an element of LEGAL.PERSONS. (Delineations are discussed in Chapters V and VI.) The candidate for the agt position is C of space NP3. Since C is an element of COMPANIES, and COMPANIES is a subset of LEGAL.PERSONS, C is accepted. A combination such as "built by a submarine" would have been rejected.

Once VP1 and PREPP1 have passed the acceptability tests, a new space, VP2, is constructed to encode the resultant VP. This new space links node 'B' of VP1 with node 'C' of NP3 via an agt arc. This new arc is visible from space VP2 (and lower spaces in the hierarchy), but is not visible from either VP1 or NP3, leaving the components encoded in VP1 and NP3 free to combine in alternatives to VP2 if necessary.

Continuing the parse, NP2 ("a-submarine") is combined with VP2 ("was-built by a-company") to form S1, after passing tests similar to those above. The obj arc linking the constituent phrases of S1 is contained in space S1 and hence is not seen from the spaces of the

 * The use of case information is described in greater detail in Section D at the end of this chapter.

constituents NP2 and VP2. Notice that the construct "a-submarine was-built by a-company", which is encoded by S1, is a spurious interpretation of utterance components. The creation of this spurious phrase could have been avoided by strict left-to-right parsing. However, in a system for understanding speech, it may be desirable for parsing to proceed from the right or from the middle (island driving). In any case, the purpose of this presentation is to show how spurious constructions arising either from misheard words or local ambiguities are handled by the SCRs.

Using rule R4, PREP2 ("of") may be combined with NP2 ("a-submarine") to form PREPP2. The network structures that are visible from space PREPP2 do not include the (spurious) obj arc from 'B' to 'S' that lies in space S1.

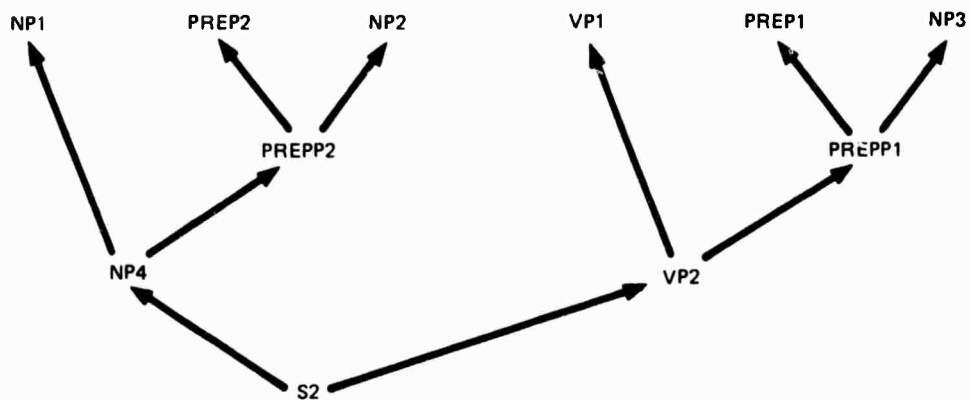
When the syntax of rule R2 suggests combining NP1 and PREPP2 to form a new NP ("a-power-plant of a-submarine"), an SCR is called. The SCR checks NP1 to see if it is relational in nature (as is "length" in "length of the Henry.L.Stimson" or "length of 425 feet") and hence expecting an argument to be supplied. Since NP1 fails this test, the SCR checks the properties of the PREP "of" and discovers that it may be used to encode HAVE.PART situations. Calling upon the delineation of HAVE.PART and appropriate surface-to-deep-case maps, the SCR determines that the HAVE.PART hypothesis provides a feasible interpretation for the NP and hence builds space NP4 with node '1' and three arcs as shown. Although these new constructs are visible from space NP4, they are not

visible from constituents NP1 and PREPP2 (and NP2). Furthermore, they cannot be seen from spurious space S1. Thus, the construction of NP4 has not altered the view of the net from S1. This is an important feature, since at this point in the processing S1 is just as likely a hypothesis as NP4. While these two hypotheses are incompatible, they are nevertheless able to share the structure of NP2 without interfering with one another.

Using rule R1, S2 is constructed from NP4 and VP2. In addition to the obj arc contained in space S2 itself, the view of the net from S2 includes all the information accessible from either space NP4 or space VP2, and hence is identical to the view from space SCRATCH of Figure VII-1. Since the parse corresponding to space S1 does not successfully account for the total input, it is rejected, and S2 is accepted as expressing the meaning of the input.

As will be described later, during the quantification phase, the structures on space S2 and those spaces that are above S2 but below KNOWLEDGE are quantified. The result of this process is exactly the SCRATCH space of Figure VII-1.

The partial ordering of spaces from S2 to KNOWLEDGE indicated in Figure VII-2 is identical to that represented more clearly in Figure VII-3 which, because of the choice of space labels, may be recognized as the parse tree of the input sentence. Consequently, the syntax of the input and the association between each syntactic unit and its



SA-3804-23R1

FIGURE VII-3 VIEWING HIERARCHY ABOVE S2

corresponding semantics have been captured in the structures built by the SCRs. As discussed below in Section C on quantification and in relation to discourse analysis in Chapter X, this association plays a central role in determining the scopes of higher-order predicates and in analyzing elliptical utterances.

2. TECHNICAL COMMENTS ON THE EXAMPLE

In the discussion of the example, a few technical points were suppressed to simplify the exposition. These will now be considered.

a. SHARING NETWORK STRUCTURES

As seen in the example, partitioning enables networks to maintain alternative hypotheses (e.g., S1 and S2) concerning the use of utterance constituents and enables such competing hypotheses to share network substructures (e.g., V2). Since partitioned structures, together with the associated feature of multiple vistas which allows alternative views of the network, make sharing so natural and straightforward, it is worthwhile to reflect upon the problem of sharing that arises in unpartitioned networks.

The root of the problem is that networks, unlike simpler list structures, are cross-linked by two-way pointers. To see the distinction, let X be some S-expression and let L1 and L2 be two list structures that contain pointers to X. In establishing pointers from L1 and L2 to X, no change is made in X itself. In particular, the creation of a pointer to X does not result in the creation of an inverse pointer from X. So both L1 and L2 may point to X without any complications since X does not point back to either structure.

In ordinary networks, the situation is different. If arcs are established to (or from) a node Y from (to) other nodes N1 and N2, then pointers are established in N1 and N2 that point to Y and pointers are established in Y that point to N1 and N2. Now, if N1 and N2 are alternatives, the following problem arises. By taking alternative N1, the structures pointed to by N1 must be taken also

(since they form a part of the extended meaning of N1). In particular, Y must be taken, and in turn the structures pointed to by Y. But this includes N2, which, being the other alternative, was to have been excluded. The point is that N1 and N2 become linked when they attempt to share the same substructure Y. This contrasts with the list structures above in which L1 and L2 could both point to X without establishing a path from L1 to L2.

There are two solutions to the sharing problem in networks. The first is not to share at all. That is, all structures that would have been shared are instead copied. This solution is expensive and, ultimately, unworkable, since there is usually some path between just about any two nodes in the network. The other solution is to establish a bookkeeping procedure that will maintain a number of different points of view. For each point of view, the bookkeeping must indicate exactly which network structures are to be included and which are not. Such bookkeeping, of course, constitutes a type of network partitioning.

b. SYNTACTIC ORDER

The visibility hierarchies shown in Figure VII-3 appear to maintain the syntactic order of constituents of a phrase. For example, these figures show that NP4 and VP2 are immediately above S2 and seem to indicate that NP4 is to the left of VP2. In reality, the orthodox vista of S2 is guaranteed to contain both space NP4 and space

VP2, but the vista itself says nothing about their relative order. Therefore, any space created by an SCR that has more than one direct parent will have a property, called PARENT.ORDER, indicating the left to right order of its parents. For example, the PARENT.ORDER of S2 is (NP4 VP2). Because of quantification considerations, the order is reversed if one of the parents is a prepositional phrase. This result reflects the quantification rule that "the scoping power of a higher-order predicate decreases from left to right except when embedded in a prepositional phrase."

c. CONCEPTUAL SPACES

Actually, it is not necessary for the system to create new spaces for all syntactic units (even though it could). For example, the spaces PREP1, PREPP1, PREP2, and PREPP2 of the example exist only conceptually. The space really associated with the syntactic unit PREPP1 is NP3. This reflects the fact that, in isolation, the prepositional phrase determines no more network structure than the NP alone.

However, empty spaces sometimes are created by the SCRs. Typically, this occurs when a new phrase has the same unquantified network structure as one of its constituents but differs from the constituent because of quantification. A new space is created in order to attach a quantification property that is to belong to the new phrase but not to the constituent.

d. COMMUNICATING WITH THE SYSTEM EXECUTIVE

Each SCR has its own set of input parameters. Typically, these include the semantic interpretations of phrase constituents and a few syntactic attributes. If any constituent is ambiguous, multiple unambiguous calls are made to SCRs. If the results of an SCR are ambiguous, a list of interpretations is returned. The interpretations returned by the SCRs are typically communicated by pairs of the form (node . vista). For example, the interpretation of the VP "was-built by a-company" is passed to the executive by the pair

(('B' . (VP2 VP1 [PREPP1] [PREP1] NP3 KNOWLEDGE)).

(Recall that [PREPP1] and [PREP1] are only conceptual.) Discounting the KNOWLEDGE space, the vista of a pair includes all spaces upon which structures have been created to encode the unquantified interpretation of the phrase. (As will be seen in a subsequent example, this vista sometimes consists of the KNOWLEDGE space alone.) The node of the pair is the so-called "head node" of the structure. It is at this node that the structure will typically be joined to other structures in creating larger phrases.

For simplicity of writing, a node-space pair may be used to represent the node-vista pair in which the vista is the orthodox vista of the space. Thus, the pair above may be abbreviated as

(('B' . VP2) .

Using this notation, the results of the various calls to the SCRs may be summarized as follows:

PHRASE	INTERPRETATION
NP1	('P' . NP1)
NP2	('S' . NP2)
NP3	('C' . NP3)
NP4	('P' . NP4)
PREP1	pre net
PREP2	pre net
PREPP1	('C' . NP3)
PREPP2	('S' . NP2)
S1	('B' . S1)
S2	('B' . S2)
VP1	('B' . VP1)
VP2	('B' . VP2)

3. INTERACTING WITH DISCOURSE: DETERMINED NOUN PHRASES

The example utterance considered above was carefully constructed to avoid complications arising from quantification or from interactions with the discourse component of the speech understanding system. However, both quantification and interaction with discourse have major impacts on the semantic aspects of the translation process. By considering a second example sentence

"General Dynamics built the American submarine,"
new facets of the semantic system, particularly its interaction with discourse, may be highlighted while still avoiding the complications of quantification.

The principal distinction between the first example sentence and the current sentence is that the former contained only indefinitely determined noun phrases whereas the latter contains definitely determined NPs. Thus, the first example concerns "a company," but the

current example concerns the particular company "General Dynamics." Likewise, the first example concerns "a submarine," but the current example concerns a particular submarine that is designated as "the American submarine."

The phrase "the American submarine" is intended for use in a context in which the partial descriptor "an American submarine" is sufficient for distinguishing a particular individual. Assuming that the current conversation concerns the Henry.L.Stimson, which is an American submarine, and the Churchill, which is a British submarine, then the phrase "the American submarine" designates the Henry.L.Stimson, and the example sentence is equivalent to the following:

"General Dynamics built the Henry.L.Stimson."

The actual task of relating "the American submarine" to the Henry.L.Stimson is performed by the discourse component and is described more fully in Chapter IX. As will be shown in this section, SCRs create the network descriptions that discourse uses in finding the referents of determined noun phrases. Further, once a referent is found, SCRs are used to incorporate the particular individuals returned by discourse when building larger structures.

Turning now to the details of translating the example sentence, the target structure which is to be produced by the parsing process is shown in Figure VII-4. Note that the SCRATCH space contains only the building node and its associated arcs. These elements constitute the new information conveyed by the sentence. Both

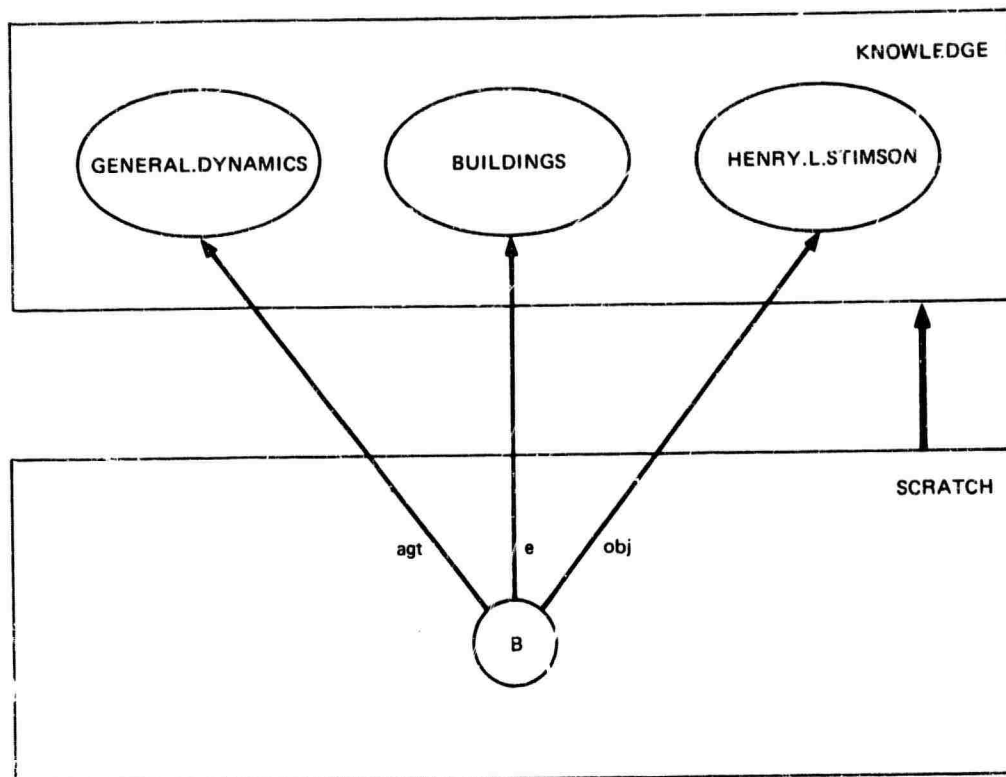


FIGURE VII-4 CONTEXT DEPENDENT PARSE TARGET STRUCTURE FOR
"GENERAL.DYNAMICS BUILT THE AMERICAN SUBMARINE"

"General.Dynamics" and "the American submarine" (i.e., "the Henry.L.Stimson") were already known to the system.

To perform the translation with minimal syntax, assume the following simplified language definition:

GRAMMAR

- R1: S => NP VP
- R2: VP => VP NP
- R3: NP => N
- R4: NP => ART MOD N

LEXICON

N: General.Dynamics, submarine
 VP: built
 MOD: American
 ART: a, an, the

Using this language definition, the parsing process will produce the various subphrases shown in Figure VII-5. The semantic and discourse interpretations of these phrases are indicated by node-space pairs that refer to the network of Figure VII-6. Figure VII-6 shows a portion of the KNOWLEDGE space and all network structures that are produced in the translation of the example sentence.

PHRASE	ENGLISH EXPRESSION	INTERPRETATION
-----	-----	-----
ART1	the	--
MOD1	American	('P' . MOD1)
N1	General.Dynamics	('General.Dynamics' . KNOWLEDGE)
N2	submarine	('S' . N2)
NP1	General.Dynamics	('General.Dynamics' . KNOWLEDGE)
NP2	the American submarine	('S' . NP2-S)
		from semantics
		('Henry.L.Stimson' . NP2-D)
		from discourse
VP1	built	('B' . VP1)
VP2	built the submarine	('B' . VP2)
S1	General.Dynamics built	
	the American submarine	('B' . S1)

Figure VII-5. NODE-SPACE PAIRS FOR PHRASES IN "GENERAL.DYNAMICS BUILT THE AMERICAN SUBMARINE"

The first word of the sentence is the N "General.Dynamics". When this word is identified in the acoustics, an SCR is called to construct an interpretation. Typically, SCRs build new network

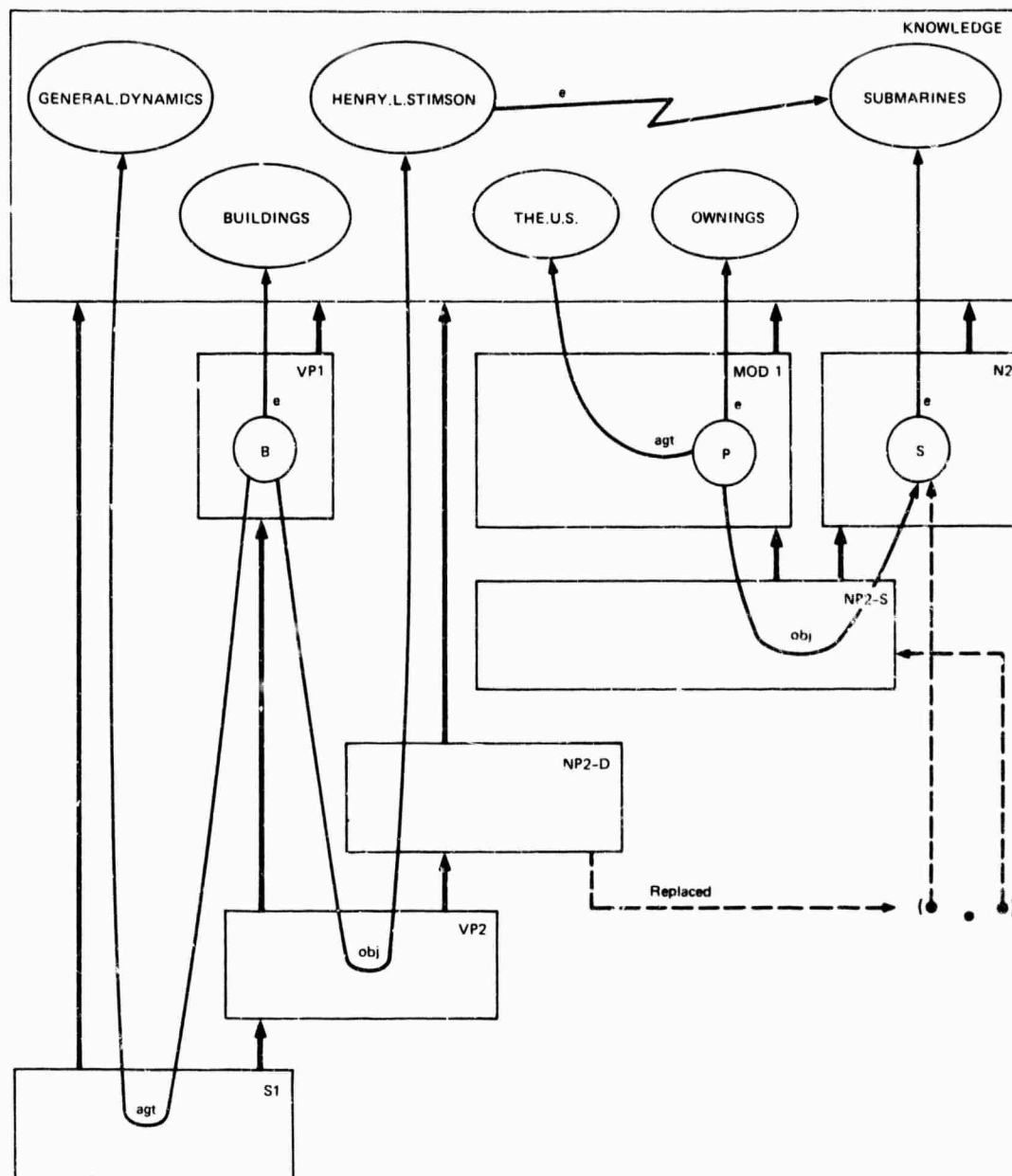


FIGURE VII-6 SCRATCH SPACES FOR "GENERAL.DYNAMICS BUILT THE AMERICAN SUBMARINE"

structures, but since information in the lexicon indicates that "General.Dynamics" has a unique referent, which is modeled by the node 'General.Dynamics' of the KNOWLEDGE space, no new network structures need be created. Rather, the SCR simply returns the node-space pair ('General.Dynamics' . KNOWLEDGE).

It has been stated previously that the interpretations produced by the semantics system indicate the association between each phrase of an utterance and its network translation (if any). This association is usually indicated by spaces. Since phrases typically cause new structures to be constructed, and since these structures usually involve more than a single node or arc, spaces (or vistas) are used to encircle the collection of structures created for the phrase. But spaces need not always be used. Since "General.Dynamics" already has a representation in the KNOWLEDGE space, no new structures are created. Further, since "General.Dynamics" is represented by a single node, a new space is not needed to bundle together a number of network structures. As seen in Figure VII-5, the semantic interpretation of "General.Dynamics" is designated by the node-space pair ('General.Dynamics' . KNOWLEDGE). In general, wherever the interpretation of a phrase is expressed by a pair in which the node is in the KNOWLEDGE space, the network translation of the phrase is simply the node of the pair.

By applying rule R3 of the grammar, the N "General.Dynamics" may be generalized to an NP. The SCR associated with this

transformation is an identity function. That is, the interpretation of the NP is the same as the interpretation of the N.

It is worth emphasizing that the SCR that associates the N "General.Dynamics" with the node 'General.Dynamics' makes this association because the lexicon indicates that the node is a unique referent that is independent of context. That is, there is only one General.Dynamics and hence the interpretation of the N "General.Dynamics" does not vary with context. A proper (and hence definitely determined) noun such as "John" is typically used to refer to some particular object, but since there may be many Johns, the object referred to depends upon the context in which the noun is used. Finding the referents of determined noun phrases with respect to context is the task of discourse and will be illustrated shortly.

Moving to the next word in the sentence, an SCR is called when the VP "built" is recognized. As was the case for the previous example sentence, this SCR creates a new space, VP1, to encode the interpretation of the VP. Within this space, a node 'B' with an e arc to BUILDINGS is created to represent a building event.

The recognition of the word "the" as an ART (article) leads to no significant processing by SCRs.

The next word, "American", has only one meaning in the speech understanding system: "owned by the U.S." (Were other interpretations allowed, the following analysis would simply be one among many.) The

SCR that is called to build an interpretation for this MOD (modifier) creates the space MOD1 shown in Figure VII-6. Within this space, a node 'P' is created to represent the ownership situation and an agt arc is created from 'P' to 'The.U.S.' to indicate that the U.S. is the owner. These structures are built in accordance with information taken from the lexical entry for "American". The lexical entry also indicates that the thing to be modified by this MOD must fill the obj case of the OWNINGS situation. The obj case is said to be the "open" case of the MOD.

The last word, "submarine", causes a space N2 to be created, upon which lies a node 'S' with an e arc to 'SUBMARINES'.

Consider now the application of rule R4 to produce the NP phrase "the American submarine" from the phrases ART1, MOD1, and N2. The SCR associated with this rule tests to see if the head of the N phrase (represented by node 'S') is a feasible candidate to fill the open case of the MOD. To do this, the SCR uses the delineation of set OWNINGS, as described previously. Once this test is satisfied, an obj arc is created from 'P' to 'S' on a new space NP2-S. This space, in combination with the spaces MOD1 and N2 of its orthodox vista, encodes the semantic interpretation of the phrase. Specifically, it describes what may be paraphrased as "A submarine that is owned by the U.S." Note particularly that it does not really represent "THE submarine that is owned by the U.S."

If the ART of the production were "a" or "an", then the description of an American submarine that is produced by the SCR would be an appropriate final interpretation of the new NP. But, since the ART is in fact "the", the description produced by semantics is only the first step in the process of properly interpreting the NP. The presence of "the" signals that the NP is definitely determined. In terms of the system, this result means that the description built by the SCR should be sufficiently specific to uniquely identify some particular object that is currently in context. It is the task of the discourse component to use the description built by the SCR to find this object.

(Note: There are other meanings conveyed by definitely determined NPs, and discourse must determine which is intended. For example, there is the generic meaning, as in "the dog is man's best friend." Also, the context is sometimes universal as in "the moon is full." With no context, "the moon" refers to the moon of earth, but in a special context might refer to one of the moons of Mars. The resolution of a determined NP sometimes depends upon the context defined by the embedding sentence itself, as in "the composer that I like best is Bach." But the case under consideration, the case in which discourse looks for an object in local context, is the only important case for the current SRI speech understanding system.)

Assuming that both the Churchill and the Henry.L.Stimson have been mentioned recently, the discourse system will determine that "the American submarine" is the Henry.L.Stimson. At a technical level,

discourse creates an interpretation that is expressed by the node-space pair

(`'Henry.L.Stimson' . NP2-D`).

Paralleling (`'General.Dynamics' . KNOWLEDGE`), since `'Henry.L.Stimson'` is on the KNOWLEDGE space, the final network translation of the NP "the American submarine" is the single node `'Henry.L.Stimson'`. The space NP2-D contains no structure. Its purpose is simply to hold the place of the space NP2-S that was created by semantics. Subsequent compositions that would have used space NP2-S will now use NP2-D. To show that NP2-D has replaced NP2-S, the property list of NP2-D contains a REPLACED property whose value is (`'S' . NP2-S`), the displaced semantic interpretation. Property lists and LISP pointers are shown in Figure VII-6 by dashed arrows.

The next rule to be applied is R2, which indicates that a new VP may be created from VP1 ("built") and NP2 ("the American submarine"). The SCR for this rule uses the surface-to-deep-case map associated with the lexical entry for "built" in determining that the syntactic direct object should map onto the deep obj case. The delineation of BUILDINGS is then used to test the feasibility of the `Henry.L.Stimson` filling this case. When this test is passed, an obj arc from `'B'` to `'Henry.L.Stimson'` is created or a new space "VP".

Similarly, rule R1 is used to establish an agt arc from `'B'` to `'General.Dynamics'` on new space S1, completing the parsing process. The quantification phase then transforms the interpretation of the total

sentence into the network of the SCRATCH space of Figure VII-4. The structures on this SCRATCH space are exactly those structures that lie on the spaces of the orthodox vista of S1, omitting space KNOWLEDGE.

4. OTHER ASPECTS OF THE SCRS

With the exception of quantification, the examples presented above have introduced all of the major aspects of the semantic composition routines. However, a few of the less central aspects are worth mentioning.

a. MULTIPLE NODE LEXICAL ENTRIES

All of the lexical items presented thus far have produced at most one node in the scratch spaces used by the SCRs. However, the MOD "American" may really be thought of as a two-node lexical item since it designates both an element of OWNINGS and fills the deep agt case of the owning situation with The.U.S.

Some relational nouns are entered in the lexicon as two node entries. For example, the interpretation of the isolated noun "beam" is shown in Figure VII-7. This word imports both the concept of a length L and the concept of this L filling the measure case of a HAVE.BEAM situation H. Since "beam", "draft", and "length" all designate elements of LENGTHS, it is the situation part that distinguishes these words from one another.

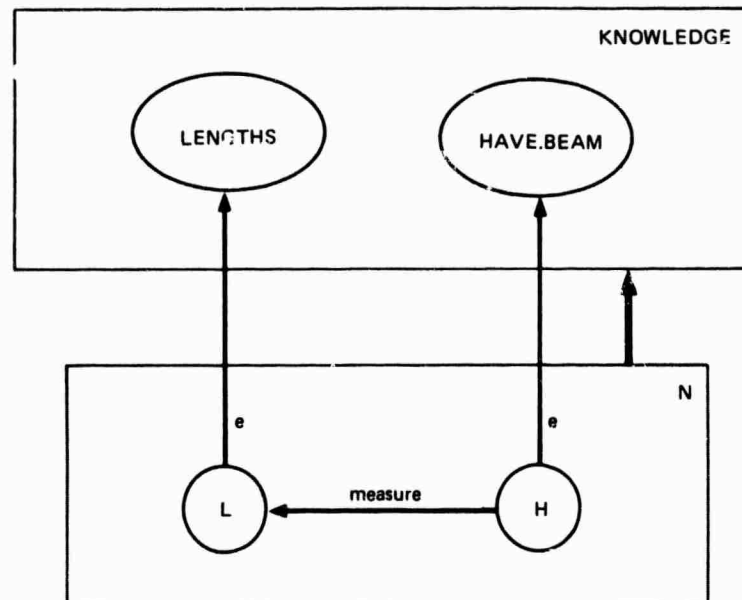


FIGURE VII-7 THE TWO NODE INTERPRETATION OF "BEAM"

b. EQUIV ARCS

In the network representing the domain model, if two nodes N1 and N2 are known to represent the same thing, then N1 and N2 are the same node. (Note carefully that two different nodes may still represent the same object because the objects represented by the nodes are not KNOWN to be equivalent -- even though they are equivalent.) In processing inputs, objects that have been described differently are often asserted to be equivalent. To show this equivalence, the objects may be connected by an equiv arc. The direction of the equiv arc is irrelevant.

For example, consider the sentence

"The Henry.L.Stimson is a ship."

The scratch spaces created in translating this sentence are shown in Figure VII-8. The first NP of this sentence ("The Henry.L.Stimson") is interpreted as designating node 'Henry.L.Stimson' of KNOWLEDGE. The second NP results in the creation of a node 'X' to represent some element of SHIPS. The copula "is" links these two concepts by asserting that they are equivalent. That is, some ship X is equivalent to the Henry.L.Stimson. Using this identity, the Henry.L.Stimson must itself be an element of SHIPS. The result of eliminating the equiv arc and substituting 'Henry.L.Stimson' for 'X' is shown in Figure VII-9.

As described in Chapter XII, the deduction component of the speech understanding system makes use of equiv arcs. In particular, an equiv arc is matched against a dummy entity, and the from- and to-nodes of the arc are paired.

5. MORE ON DELINEATIONS

Above, in processing example sentences, delineations were used as semantic tests to determine whether a given object could play a given role in a situation of a given type. In a speech understanding system, the principal contribution of such tests is to throw out spurious combinations which, typically, are proposed as a result of misheard words. But these delineation tests can also clear up certain ambiguities.

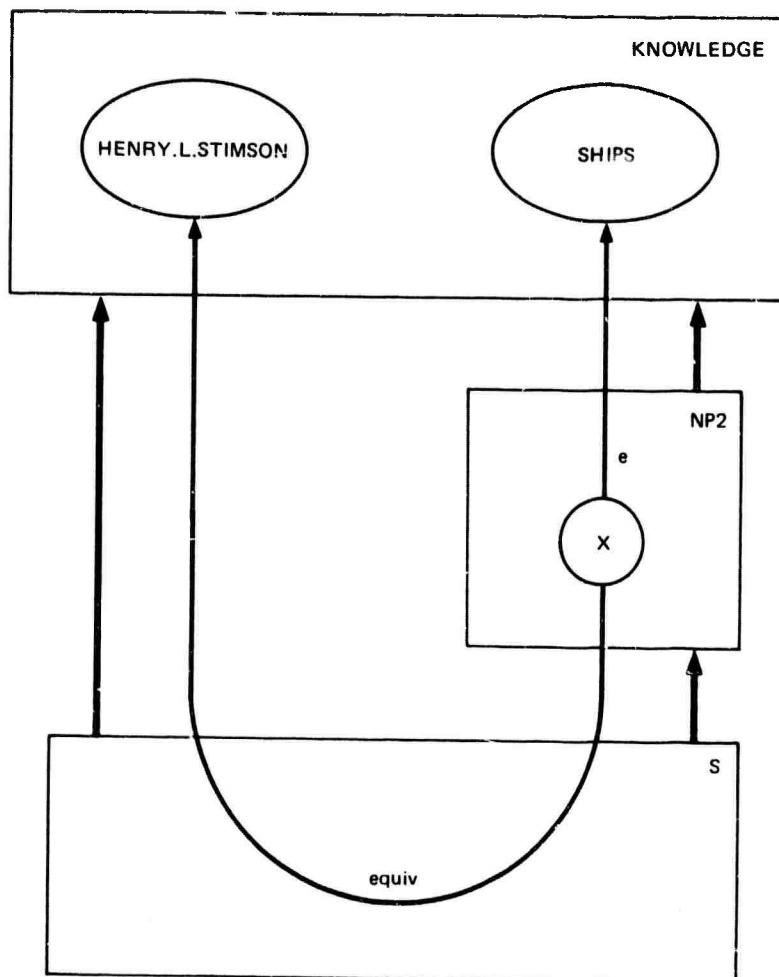


FIGURE VII-8 SCRATCH SPACES WITH EQUIV ARC FOR "THE HENRY.L.STIMSON IS A SHIP"

Given the skeletal sentence

"X built Y",

it is clear that, if the sentence makes sense at all, X designates the deep agt of a building event and Y designates the deep obj. However, for the skeletal noun phrase,

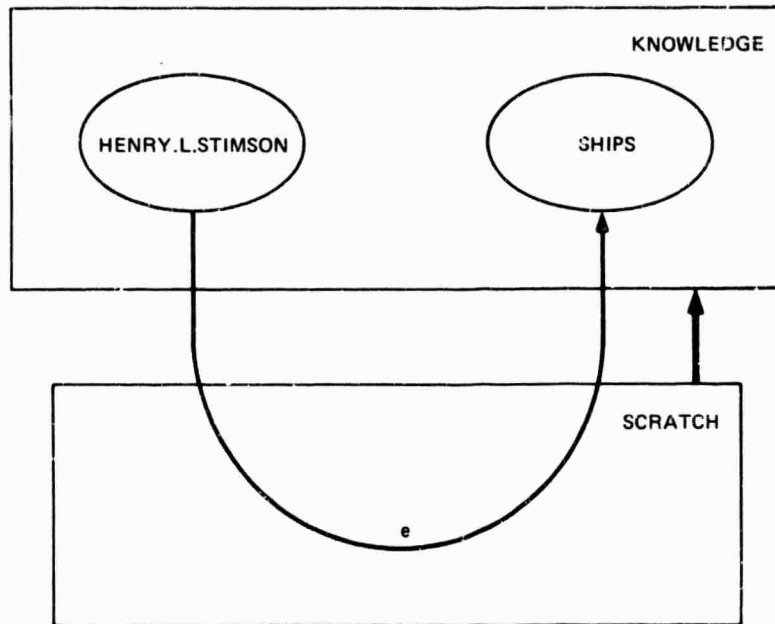


FIGURE VII-9 SIMPLIFIED INTERPRETATION OF "THE HENRY.L.STIMSON IS A SHIP"

"beam of Z",

the Z might be either the obj of a HAVE.BEAM relationship (as in "beam of the Henry.L.Stimson") or the measure of such a relationship (as in "beam of 33 feet"). The delineation of HAVE.BEAM, by indicating that the obj must be a physical object and the measure must be a length, provides sufficient information to determine the role played by the Z.

6. SEMANTIC COMPOSITION RULE SUMMARY

The operations of the semantic composition rules may be summarized as follows. When a content word is identified in the acoustic stream, an SCR is called to interpret the word. The

interpretation structure is a network fragment that describes the input by relating it to concepts in the KNOWLEDGE space (which encodes the system's domain model). Information concerning how to interpret a given isolated word comes largely from the lexicon.

Information concerning if and how subphrases may be combined to form larger units is encoded procedurally in the various SCRs and declaratively in the delineations of sets.

For determined noun phrases, the indefinite interpretations constructed by the SCRs are typically replaced by discourse interpretations referencing particular items in the domain model. The interpretations built by SCRs are used by the discourse system in finding the appropriate referent. The interpretations from the discourse system may be combined into larger combinations in the same manner as the interpretations created by the SCRs themselves.

The network interpretation of a total utterance is divided among a number of scratch spaces. Each syntactic unit of the input may be identified either with a single node on the KNOWLEDGE space or with some set (i.e., vista) of these scratch spaces. The syntactic phrase structuring of the input is reflected in the vista hierarchy relating the various spaces.

In addition to the node and arc structures, the various spaces created by SCRs may carry information on their property lists concerning quantification. The nature of this information and its role in completing the translation process are the subjects treated next.

C. PHASE II: QUANTIFICATION

The quantification procedure performed by the semantic component and described in this section is the final step in the construction of a structure representing the meaning of an input. This operation is performed on the (as yet unquantified) interpretation structure of a complete utterance that was produced either by an SCR (in the case of a complete sentence) or by the discourse component (in the case of elliptical expansion).

The task of the quantification phase is to introduce higher-order predicates and their associated scopes into the structure produced by the composition phase. Although the scoping of all higher-order predicates is performed during this process, the procedure is called the "quantification phase" because most of the higher-order predicates that have been considered are, in fact, quantifiers.

The introduction of scopes into the translation net is delayed until a postphase because of the highly context-sensitive nature of scope determination. While quantifiers are frequently indicated within noun phrases, their influence is generally not confined to the noun phrase itself, but rather is brought to bear on higher constituents in which the noun phrase is embedded. Furthermore, the scopes of most English quantifiers are affected by other quantifiers that appear in the utterance. The interaction of quantifiers is influenced by the syntax of the higher level constituents that incorporate them, by the relative

scoping strengths of the actual English words used, and by the physical feasibility of readings.

The quantification process is largely performed by repartitioning the nodes and arcs of the unquantified interpretation structures. That is, while leaving the original partitioning in place to save the syntactic history of the input for discourse analysis (i.e., for the expansion of future elliptical inputs), nodes and arcs are placed in new spaces that are ordered hierarchically to indicate the nesting of higher-order predicates. For example, a typical new space might define the boundaries of the scope of a universal variable. In addition to repartitioning, some new nodes also may be introduced into the translation structure to represent such logical connectives as IMPLICATIONS and NEGATIONS.

Information for deciding how to define the new "quantification" spaces and how to order the spaces is taken from the network structures built earlier in the semantic composition phase. In particular, the syntactic structure of the input, as shown in the hierarchy of scratch spaces, plays an important role. Heavy reliance is also placed on information (not yet discussed) that is encoded on the property lists of the scratch spaces.

The approach to determining the scopes of quantifiers that is presented in this section has been influenced by the game theory technique advanced by Hintikka (1974). However, certain engineering

expediciencies have been used in applying portions of his technique to an operational system based on partitioned semantic networks.

1. OVERVIEW

To gain a perspective on the overall quantification procedure, consider the processing of the example query

"Did General.Dynamics build every Lafayette?"

(The Lafayettes compose one class of nuclear submarines.) The scratch spaces for this query that were built by the SCRs during the composition phase are shown in Figure VII-10.

Simply put, the node and arc structure created by semantic composition is the same as would have been created for

"General.Dynamics built a Lafayette."

However, two of the spaces have special properties relating to quantification (or, more generally, higher-order predicates). Space NP3 includes information about the English quantifier "every" and space S1_h includes information about the special YN (i.e., yes/no) quantifier."

In more detail, the noun phrase "General.Dynamics" is interpreted by the single node 'General.Dynamics' of the KNOWLEDGE space. The VP "built" results in space VP1, as in previous examples. Also paralleling previous examples, the noun "Lafayette" results in a node 'L' with an arc to 'LAFAYETTES' being created in a new space N2. The quantifier "every" is combined with this noun to form a new noun

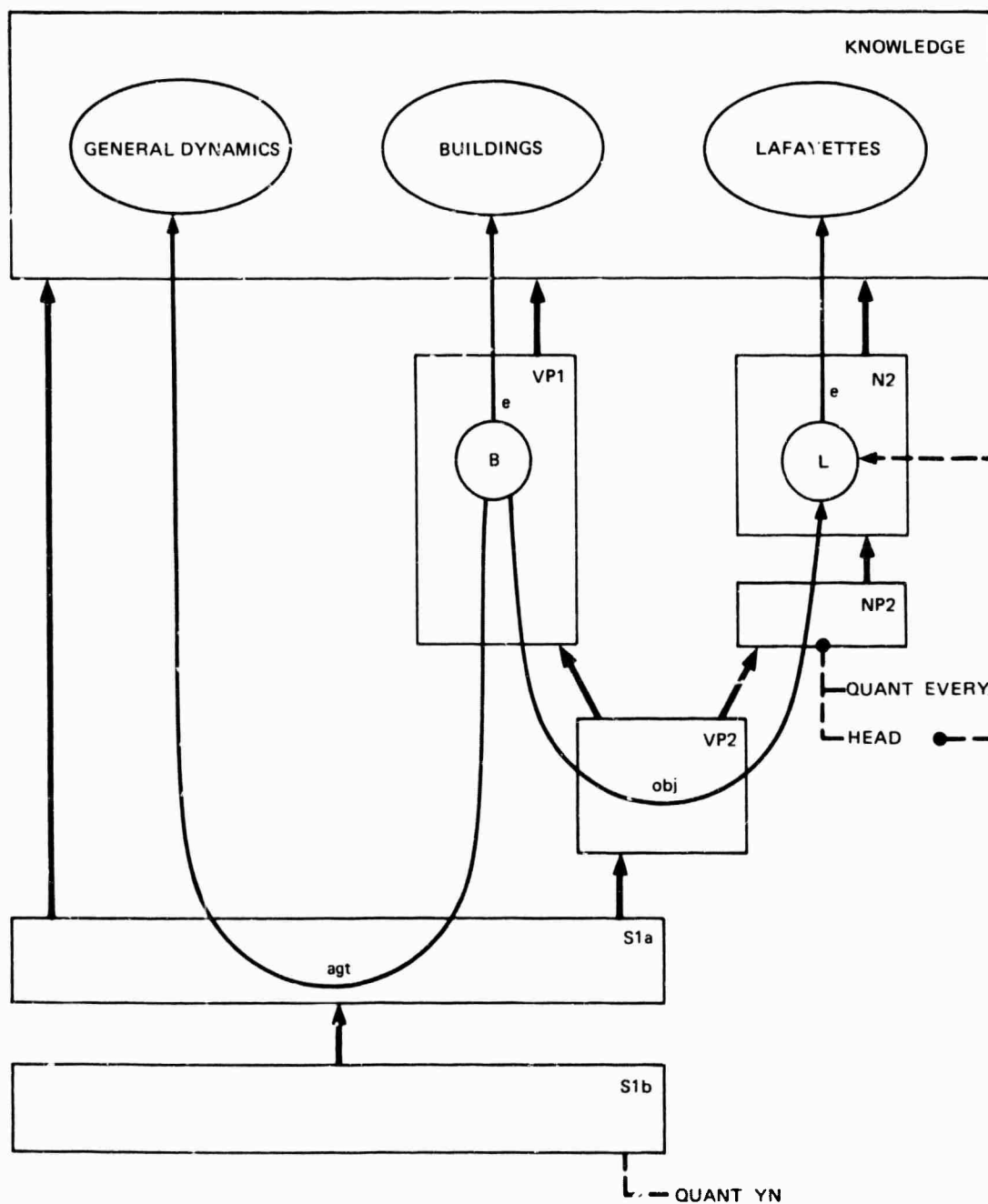


FIGURE VII-10 SCR SPACES FOR "DID GENERAL.DYNAMICS BUILD EVERY LAFAYETTE?"

phrase, "every Lafayette." This NP is represented by the vista from new space NP2. The view from NP2 inherits the noun structure of N2. But, through the property list of space NP2, there is added to this noun structure the information that the English quantifier "every" was used in the formation of the NP. The empty space NP2 is created solely to provide a place to attach this property list. If the properties were hung from space N2, then the interpretation of the noun in isolation would be altered and would not be available for incorporation in alternative hypotheses. (There may, for example, be a hypothesis in which the acoustic signal preceding "Lafayette" is interpreted as containing some word other than "every.")

VP1 and NP2 are combined to form VP2, "built every Lafayette," in a fashion analogous to previous examples. In the last stage of composition, the AUXD (the auxiliary verb of the "do" family) "did" is combined with the NP "General.Dynamics" and VP2 to form a complete query. The building operations for node and arc structures are the same as if the NP and VP were being combined to form an assertion. But the pattern

AUXD NP VP

signals that the validity of the assertion

NP VP

is to be tested by a YES/NO query. Since a test involving a proposition is second-order, the request for a test is not translated into a network structure by the SCR. Rather, the creation of a (second-order) network

structure to encode the YES/NO query involving the proposition NP-VP is delayed until the quantification phase. To indicate that this structure is to be built, the space S1b is created below S1a and is marked by the property value pair (QUANT . YN). (Space S1a with its vista is the interpretation of proposition NP-VP.) In general, a QUANT property is assigned to a space whenever a quantification or other higher-order structure must be built during the second phase of semantic processing.

Using the structure built during the composition phase as its sole input, the quantification phase builds a network structure that provides a complete (literal) interpretation of the original input utterance, which includes all higher-order structures. For the example query, the structure of Figure VII-10 is transformed into the structure of Figure VII-11 (while the structure of Figure VII-10 is preserved for subsequent discourse analysis).

Space T of Figure VII-11 is the target translation space for the query. At the top level, T encodes a request R for information of the YES/NO type. (See the discussion of REQUESTS.YN in Chapter V, Section E.3.a, and Chapter XI, Section B.1.) The proposition of this request, whose validity determines the answer to the query, is encoded on space P. In particular, P encodes the proposition

"For every L, if L is an element of LAFAYETTES,
then L was built by General.Dynamics."

Note that this proposition contains the universally quantified variable L and that the quantification is encoded in accordance with the techniques described in Chapter V, Section E.2.

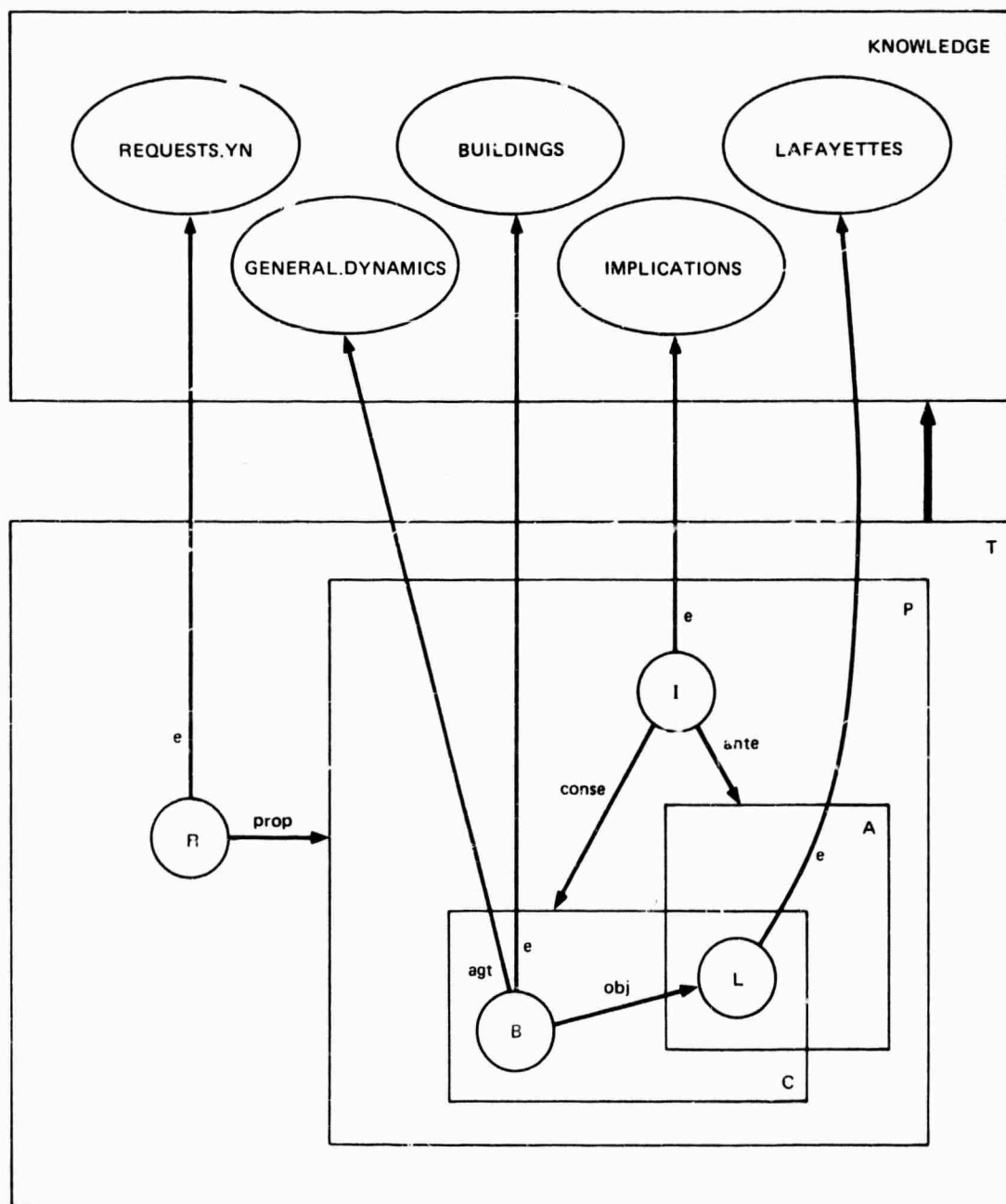


FIGURE VII-11 ULTIMATE TRANSLATION OF "DID GENERAL.DYNAMICS BUILD EVERY LAFAYETTE?"

The conversion of a semantic composition structure (i.e., the final product of the SCRs) into a fully quantified translation is performed by applying "quantification" functions (Q functions) to the various spaces created by the SCRs. Exactly one Q function is applied to each space, and the process is completed when the last space has been processed. The Q functions that are applied determine the types of higher-order structures that are produced, and the order in which spaces are selected for the application of a Q function determines the nesting of predicate scopes. Only a small number of Q functions are used.

The particular Q function to be applied to a given space is determined solely by the QUANT value of the space. (Spaces with no QUANT property are understood to have the value NIL.) The order in which the functions are applied is determined by calculations involving the syntax of the utterance and the relative scoping strength of those quantifiers that occur in the semantic composition structure.

Omitting the details of the order of space selection, the conversion of the example query proceeds as follows: First, an empty translation space T is created below the KNOWLEDGE space. This space is designated as the current active space for the creation of new structures by subsequent calls to Q functions. Then, space S1b is selected and the value of its QUANT property is mapped onto the Q function Q.YN. Q.YN is then applied to space S1b.

Q.YN builds structures to represent a YES/NO request on the active translation space, T. In particular, the structures of space T shown in Figure VII-12 are created by Q.YN. These structures consist of a node 'R', an e arc from 'R' to 'REQUESTS.YN', a new supernode 'P', and a prop arc from 'R' to 'P'. Upon completing this structure building operation, Q.YN designates the new space P as the active space for the application of Q functions to any space above S1b in the viewing hierarchy. For the current example, this includes all other spaces created during the composition phase.

The next space that is selected is space NP2. After the value of its QUANT property has been mapped onto the function Q.UNIV, Q.UNIV is called to build structures in the currently active translation space, P. The structures built by this call to Q.UNIV are shown in Figure VII-13. These include an implication node 'I' with its corresponding ante and conse spaces, A and C. After creating these structures, Q.UNIV copies node 'L' from composition space N2 onto both A and C. Q.UNIV copies 'L' because it is marked as the HEAD node of NP2. This copying, which is relatively inexpensive, does not alter the structure of N2, leaving it intact for use in discourse analysis. However, the copying does cause spaces A and C to overlap and hence, by the overlapping convention of IMPLICATIONS (presented in Chapter V, Section E.2.d), establishes L as a universal variable.

Upon completion of its building and copying operations, Q.UNIV splits the remainder of the quantification process into two

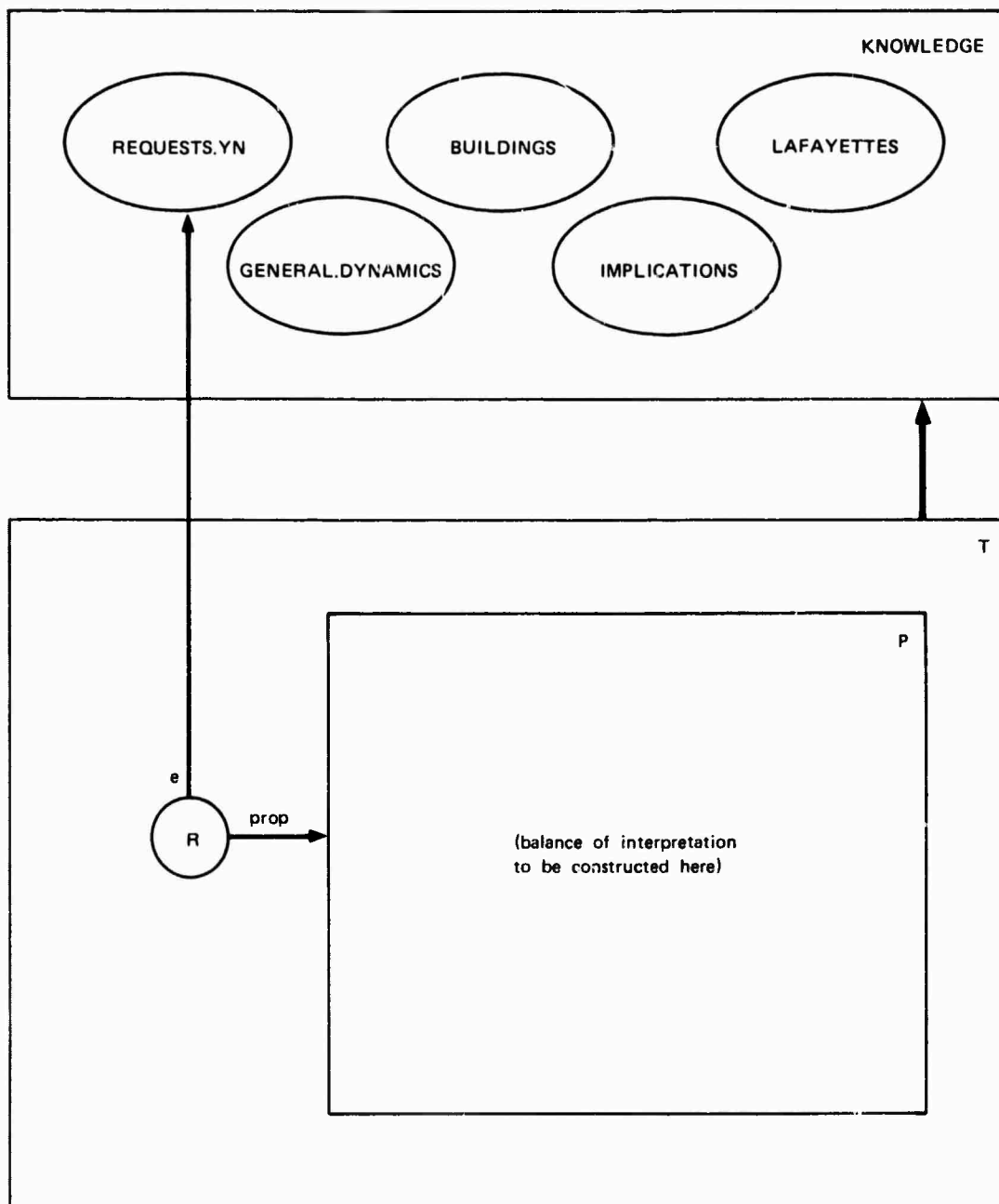


FIGURE VII-12 RESULT OF Q.YN SCOPING PROCEDURE

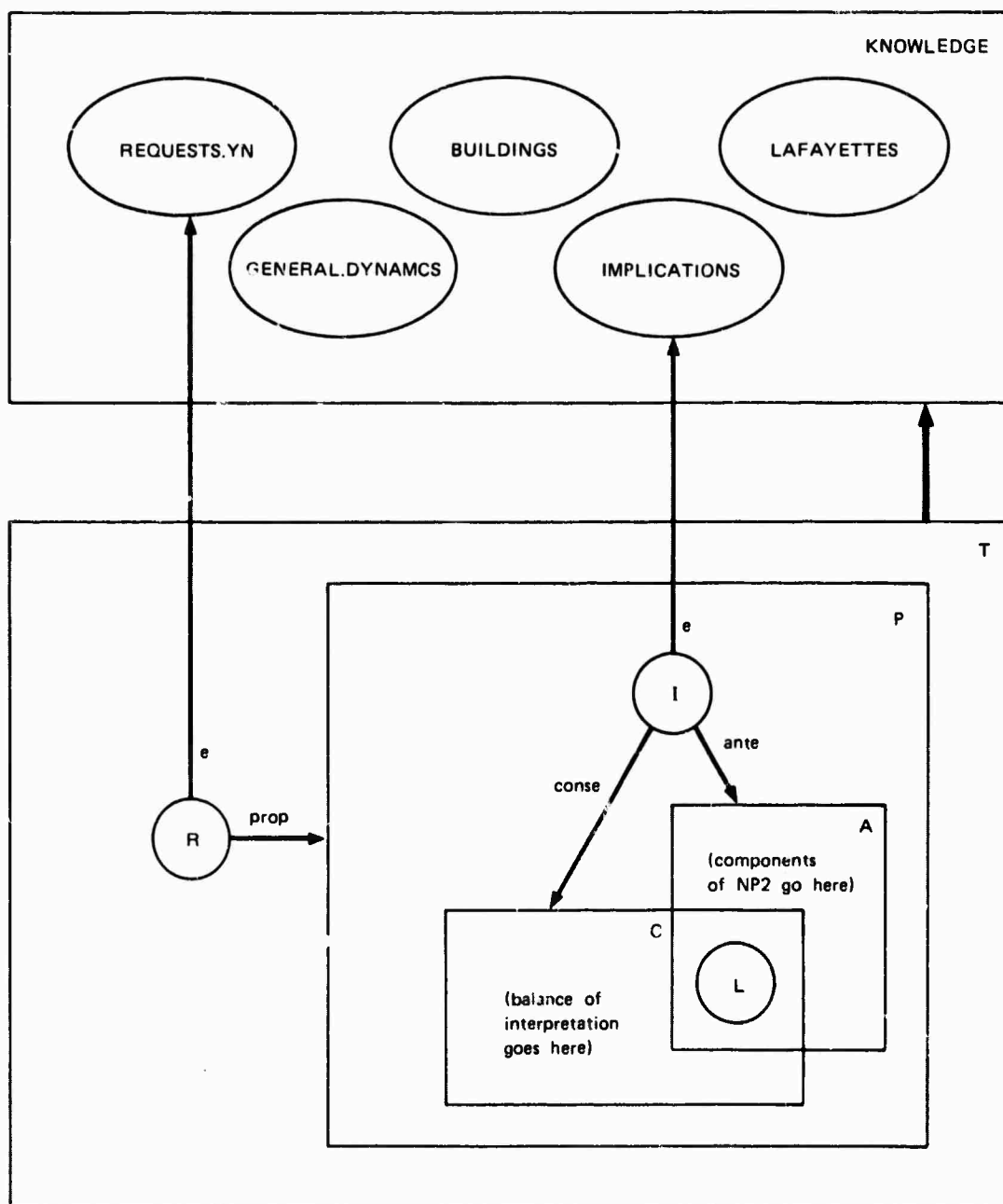


FIGURE VII-13 RESULT OF Q.UNIV SCOPING PROCEDURE

subprocesses. In the first of these subprocesses, the spaces above NP2 (the space that activated the function) are to be considered, using space A as the active space. In the second subprocess, all other spaces currently pending are to be considered, using space C as the active space.

In the first of these subprocesses, only space N2 remains to be processed. Since this space has no QUANT value, the default function Q.EXISTS is applied. This function simply copies all the structures of NP2 onto the currently active translation space, A. Since 'L' already exists on A, this copy operation acts as a no-op. But the e arc from 'L' to 'LAFAYETTES' is transferred.

In the second of the subprocesses, spaces S1a, VP2, and VP1 are considered, using C as the active space. Since none of these spaces have a QUANT property, Q.EXISTS is applied to each with the result that their structures are copied onto space C. This copying process in no way alters the structures created during semantic composition.

With S1b, VP2, and VP1 processed, all the semantic composition spaces have had a Q function applied, and the quantification phase of translation is completed.

2. THE QUANTIFIERS

A list of the quantifiers used in the system with their associated Q functions and strengths is presented in Figure VII-14.

QUANTIFIER	Q FUNCTION	STRENGTH
-----	-----	-----
ALL	Q.UNIV	2
ANY	Q.UNIV	7
BOTH	Q.UNIV	2
(CONSTANT)	--	infinite
EACH	Q.UNIV	6
EITHER	Q.UNIV	2
EVERY	Q.UNIV	2
NEITHER	Q.NO-EXIST	3
(NIL)	Q.EXISTS	0
NO	Q.NO-EXIST	3
NONE	Q.NO-EXIST	2
NOT	Q.NEG	4
PL-DEF	Q.UNIV	3
PL-NUMBERED	Q.SET	1
PL-OPEN	Q.EXISTS	0
SOME	Q.EXISTS	1
WH	Q.WH	5
YN	Q.YN	8

Figure VII-14. THE Q FUNCTIONS AND STRENGTHS OF QUANTIFIERS

(The word "quantifier," as used here means "any MARKER denoting a higher-order predicate and its strength.") The quantifiers include both English words (all, any, both, each, either, every, neither, no, none, not, and some) and quantifiers derived from structure (CONSTANT, NIL, PL-DEF, PL-NUMBERED, PL-OPEN, WH, and YN).

The quantifiers associated with English words are typically found during the composition phase by spotting constructions of the form
quantifier-word noun-structure

as in "all submarines", "every destroyer", "no torpedo launchers". The structural quantifiers are found by noting the structure of inputs. For example, the YN quantifier is signaled by the construction

is X Y?

Whenever a quantifier is found in the composition process, it becomes the value of the QUANT property of one of the composition spaces.

Each quantifier denotes two separate pieces of information: a higher-order predicate and the relative scoping strength with which that predicate is to be used. In the speech understanding system, a different Q function is used for each of the higher-order predicates recognized by the system, and it is this Q function, rather than the predicate itself, that is directly associated with a quantifier.

The interpretations placed on the English word quantifiers are quite straightforward. ALL, ANY, BOTH, EACH, EITHER, and EVERY are all interpreted as denoting universal quantification. In particular, the quantifier ANY is never used in the existential interpretation. The quantifiers are only applied to count (as opposed to mass) nouns and the individual (as opposed to the collective) interpretation is always assumed. Thus "John saw all the men" is interpreted as "For every man, John saw him" as opposed to "John saw the group of men."

The English word quantifier SOME is interpreted as denoting existential quantification. NO, NONE, and NEITHER denote "there does not exist." NOT denotes negation.

The structural quantifiers denote some of the more exotic higher-order predicates. WH (signaled by words such as "who," "what," "which," and "how many") is used to identify a WH-type request for information. Similarly, the YN quantifier denotes a YES/NO query.

The PL-DEF quantifier arises from plural, definitely determined noun phrases such as "those five ships" in "General.Dynamics built those five ships." During the composition phase, the discourse component resolves plural, determined NPs to nodes on the KNOWLEDGE space representing sets. The PL-DEF denotes a universal quantification over one of these resolved sets. For example, "for every member of the set consisting of <those five ships>, General.Dynamics built it."

PL-NUMBERED arises from plural noun phrases that explicitly designate a number (e.g., "five men", "two of the submarines", "how many engines"). This quantifier is associated with a predicate over two objects N and P that may be paraphrased as

"N is the cardinality of the set defined by the predicate P."

For example, consider the statement

"General.Dynamics built 31 Lafayettes."

During the composition phase, the phrase "31 Lafayettes" signals the PL-NUMBERED quantifier. In the quantification phase, this results in the creation of a structure encoding

"31 is the cardinality of set S, where x is an element of S if and only if x is a Lafayette and x was built by General.Dynamics."

If a plural noun phrase contains no English quantifier, no definite determiner, and no number designation, then it signals a PL-OPEN quantification. For example, "submarines" in "General.Dynamics built submarines" signals a PL-OPEN. Under normal circumstances, PL-OPEN simply indicates existential quantification. For the example just cited the interpretation would be

"There exists a submarine that General.Dynamics built."

(This interpretation loses the information that more than one submarine was built, but the structure for the more complete interpretation was considered more expensive than it was worth for the present implementation.)

Unlike any other quantifier in the system, the effect of PL-OPEN can sometimes be superseded by stronger quantifiers. In particular, if a PL-OPEN type of NP is the subject of a sentence, then an ALL quantifier is created by the sentence-level SCR to supersede the PL-OPEN. For example, in

"Ships are built by corporations"

both "ships" and "corporations" are PL-OPEN NPs. But since "ships" is the subject, an ALL quantifier at the sentence level supersedes the PL-OPEN. Thus the interpretation is

"For every ship there exists a corporation that built it."

Most spaces created during semantic composition are not marked as being quantified at all and may be thought of as having a QUANT value of NIL. This NIL value signals existential quantification. The sentence

"A power plant of a submarine was built by a company",
which was the first example considered under semantic composition, is purely existential.

Some types of phrases are mapped directly onto the KNOWLEDGE space during the composition phase. For example, phrases with unique referents such as "General.Dynamics" and (in context) "the American submarine" are so mapped. Such KNOWLEDGE space nodes are tantamount to CONSTANTS and are therefore unaffected by quantification.

3. SPACE ORDERING

The nesting of scopes by Q functions is critically dependent upon the order in which spaces are selected for Q-function application. It has been suggested that this order would best be established by game theory considerations as outlined by Hintikka (1974). However, the game theory approach did not appear suitable for immediate adaptation to a computational system. Therefore, a more engineering-oriented approach, based on syntax and quantifier strength, has been used in the SRI speech understanding system.

The ordering of spaces for Q function applications conforms to the following rule:*

GIVEN any two composition spaces,

IF either has a QUANT of greater strength than the other,
then the stronger is taken first.

* Doug Appelt and I are currently working on a revised version of this rule. Appelt has observed that when an English-word quantifier is the first word of a sentence, its strength relative to other English-word quantifiers is greatly increased. We have also observed that quantifiers in fronted adverbial phrases outscope quantifiers in the balance of the sentence, and that when an "ANY" falls within the scope of a WH or YN, it behaves existentially. Updates to the system based on these observations are pending.

OTHERWISE, if either is in the orthodox vista of the other, then the space in whose vista the other lies is taken first (i.e., compounds dominate their constituents).

OTHERWISE, that space corresponding to the logically leftmost syntactic constituent is taken first.

With one exception, a syntactic unit X is to the logical left of syntactic unit Y if X appears before Y in the sentence. The exception is the case in which X and Y combine with a preposition to form a new syntactic unit Z as in the production

$$Z \Rightarrow \text{PREP } Y .$$

For this case only, Y is considered to be to the logical left of X. (Example: "Every engine of every submarine" is roughly equivalent with "For every submarine, for every engine of that sub".)

Applying the above rule to the example input of Figure VII-10 yields the following analysis: Space S1b is the first space for Q-function application because its QUANT strength, at 8 for a YN, is greater than any other. Space NP2 comes next with a QUANT strength of 2 for EVERY. The other spaces (S1a, VP2, VP1, and N2) all have strength 0. Since all of these other spaces are in the orthodox vista of S1a, S1a comes next. Similarly, VP2 comes before VP1 or N2. VP2 comes before N2 because it is to the left of N2, and N2 comes last.

As was the case in the introductory example, the application of some Q functions may cause the quantification process to be split into multiple subprocesses, each with its own active target space. When this happens, those spaces that remain unconsidered are divided at

the subprocesses. But in each subprocess, the spaces are considered in the same order as they would have been had the subprocess remained joined.

To gain a better feel for the influence of quantifier strength over the scoping of predicates, consider the two queries

"Who built every X?"

and

"Who built each X?"

These queries are of interest because the WH quantifier signaled by the word "who" has a strength of 5, which lies between the strength of EVERY (strength 2) and EACH (strength 6).

Since WH outscopes EVERY, the interpretation of the first query goes something like this: "Who is that one builder B such that for all x in X, B built x?" Since WH is outscoped by EACH, the interpretation of the second query goes something like this: "For all x in X, who built x?" The network structures representing these two queries are shown in Chapter V, Section E.3.b.

Another pair of sentences whose interpretations differ only with respect to scope are

"All the men didn't go."

and

"Each of the men didn't go."

In the first, not all went. In the second, not any went. (Of course, "not all went" versus "not any went" is just another case in point.)

The scoping scheme as outlined above is far from perfect and should be regarded as merely a first cut at a difficult but fascinating problem.

D. THE USE OF CASE INFORMATION

Case information establishes a link between certain syntactic and semantic constructions.* It serves two purposes: First, it provides a basis for using information from a syntactic structure in determining what semantic relationships hold in a particular phrase. Second, it is a relatively simple mechanism for eliminating incorrect interpretations by rejecting unallowable semantic relationships and by blocking syntactic predictions for words that cannot possibly fit in the current semantic context. Both uses of case information have already figured in examples earlier in this chapter. Here we will describe the mechanism involved and illustrate its application.

Every word that conveys the concept of a situation has contained in its lexical entry (1) a pointer to the semantic net representation for the set of similar situations, and (2) a statement of the syntactic attributes that signal which syntactic units specify which semantic roles in situations of that type. Verbs, certain prepositions, modifiers, adjectives, and certain nouns are interpreted semantically in terms of situations and have this information associated with them in the lexicon. When a word or phrase conveying situation data is added to

* This section was prepared by Ann Robinson.

a phrase, this information is used along with the syntactic attributes of the phrase to determine the case relations allowed among particular constituents. If the phrase satisfies the relations, the information is used by the semantic composition routines to build the semantic structure representing it. The information also is used to eliminate interpretations where these relations cannot hold.

Since many verbs map the same surface constituents into the same case relations, with the only difference between verbs being the particular situation types, these verbs are grouped together and described by common paradigms (see Celce-Murcia, 1976). Thus, verbs like 'build', 'own', and 'construct' all follow a common paradigm which indicates that in the active voice, the syntactic subject fills the agt (agent) case and the syntactic object fills the obj (object) case, and that they are reversed in the passive voice. Cases may be obligatory or optional, and all the obligatory ones must be filled for a sentence interpretation to be accepted.

To see in more detail how this information is used to check possible phrases and to block predictions of words by the executive that would otherwise be made on the basis of syntactic information, it is necessary to look more closely at what information is available. As has been described in Chapter V, Section E.2.g, each situation has a delineating element that has case arcs connecting it to other nodes in the network. As a result, it is possible to determine for each case arc the set of items that can be in the case relation. Structures

indicating a particular instance of that situation must correspond to the restrictions that are specified. For example, semantic structures can not be built for phrases like 'The Henry L. Stimson owns the U.S.' because submarines cannot own countries.

To save the time that would be otherwise be required to compute for each noun whether its possible referents are included in a specific set, nouns are predivided into subcategories that correspond to the sets allowed in particular case relations. When a word that refers to a situation is added to a phrase, the information associated with that word is used along with syntactic information to determine which case relations the other constituents in the phrase can fill. If other constituents have been found and are compatible, then the phrase is built. If the other constituents have not yet been found, then the information indicating what case each constituent can fill is used to determine what subcategory or subcategories of nouns can occur in that constituent. When the executive is ready to predict nouns for each subcategory, it first checks the attributes of that subcategory in the current context. If that subcategory is not allowed semantically because of the case (or other) constraints, then that prediction is eliminated. If that subcategory is allowable then the predictions for individual words are made.

As an example, consider the sentence "Who owns the Henry L. Stimson?" as shown in Figure VII-15. The verb "owns" corresponds to the semantic situation set OWNINGS. This has two associated case arcs:

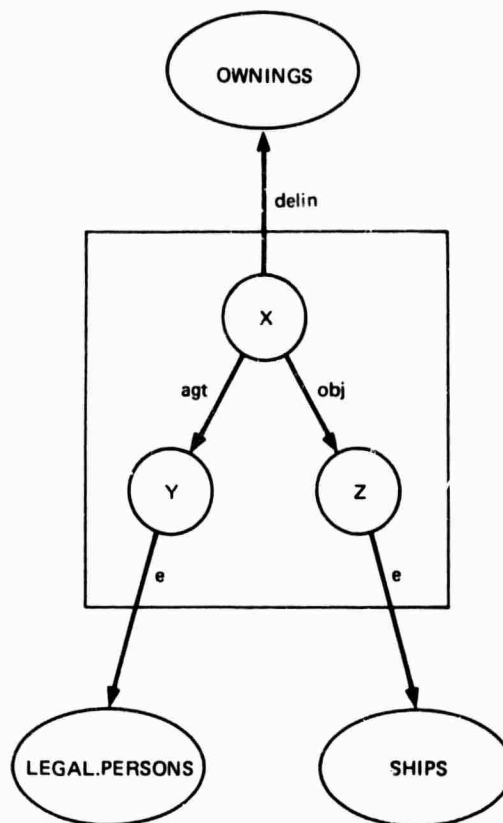


FIGURE VII-15 SEMANTIC NET REPRESENTATION
OF THE OWNING SITUATION

agt and obj. The agt arc indicates the the set of all possible owners, and the obj arc indicates the set of all things that can be owned. In this domain, the owners are all the legal persons: companies and countries. When the verb "owns" is found, and a partial verb phrase is constructed, the case information is used to restrict the nouns that can follow it. The verb "owns" is active, so the verb phrase must be of the form V NP, i.e., a noun phrase must follow the verb. The case information restricts the NP to include one of the nouns in the

subcategory of nouns that is associated with the set of ships. If any complete noun phrases have been found to the right of the verb, they will be combined with the verb only if they meet the 'case' criteria, namely, that the head noun reference a ship. If a successful interpretation has not been found, when the executive is ready to predict nouns for the NP constituent, it will first check each noun subcategory to see if that subcategory could fit at that place. In this example, the only allowable subcategory is ships. Companies, countries, measures, and the like are not allowed. The executive will only predict the individual words in that subcategory. Henry.L.Stimson is a submarine, and thus the noun "Henry L. Stimson" is in that subcategory. When the phrase "the Henry L. Stimson" is found and specified as a noun phrase, the information that it is likely to fit the obj case in the semantic representation is given to the semantic composition routines along with the semantic structures for "own" and "the Henry L. Stimson". With the VP thus completed, the information is used in the S rule to determine what possible cases the surface subject can fill. Since the obj case is filled, the remaining NP must fill the agt case. "Who" is consistent with the constraint on agt, that it be a reference to a legal person, so it is combined with the VP and the information given to the semantic composition routines to complete the semantic interpretation.

VIII DISCOURSE ANALYSIS

Prepared by Barbara J. Grosz

CONTENTS

- A. Introduction
- B. Dialog Collection and Analysis
- C. Collection of the Dialogs
 - 1. Purpose
 - 2. Task Dialogs
 - 3. Data Base Dialogs
- D. Analysis of the Dialogs
 - 1. Overview
 - 2. The Structure of the Dialogs
 - a. Task-Oriented Dialogs
 - b. The Data Base Dialogs
 - c. Kinds of Subdialogs
 - d. Opening and Closing of Subdialogs
 - e. Multiple Uses of O.K.
 - f. Multiple Open Subtasks
 - 3. Reference
 - 4. Kinds of Utterances
 - 5. Ellipsis
 - 6. Lexicon
 - 7. Descriptions
 - a. Specification
 - b. Categories of Features
 - c. Perspective
 - 8. Miscellaneous Observations

A. INTRODUCTION

Discourse processing deals with the problems arising from the necessity of relating utterances to the context in which they occur. Utterances are not spoken in isolation. Attempts to interpret an utterance in isolation often yield ambiguities that disappear once the surrounding context in which the utterance occurs is considered. Both the preceding discourse context -- the utterances that have already occurred -- and the situational context -- the environment in which an utterance occurs -- affect the interpretation of the utterance.

Development of the discourse component of the SRI speech understanding system began with the collection and analysis of several dialogs. Dialog is the most natural mode of language interaction with computers. Furthermore, from the system building and testing points of view, dialog provides the possibility of checking the system's understanding as the discourse progresses (and hence, the opportunity for clarification before errors compound themselves) and enables the system to influence the discourse. Finally, since dialog is a primary use of language, the study of dialog reveals many of the basic mechanisms in language communication.

Two kinds of dialogs were collected: a set of task-oriented dialogs involving communication between two people cooperating to complete a task; and a set of data-base-oriented dialogs involving communication directed toward obtaining information from a computer data base. The

remainder of this chapter describes the differences between these two kinds of dialog and the procedures for collecting them and gives analyses of the dialogs collected. The analyses were directed toward determining the range of phenomena present in the dialogs. In particular, we were interested in determining those characteristics of the dialogs that were amenable to formalization and incorporation in a language understanding system and in determining ways of encoding and using the information present in both the dialog itself and in the surrounding task context to aid in interpretation of successive utterances.

The dialog analysis revealed that contextual influences operate at two different levels in a discourse. First, the global context in which an utterance occurs -- the total discourse and situational setting -- provides one set of constraints on interpretation of the utterance. For example, choices between word senses are influenced by context at this level. The second set of constraints is provided by the immediate context of closely preceding utterances. A natural language-understanding system must provide for interpretation of utterances (and their parts) in terms of both of these levels of context.

The discourse component of the speech understanding system keeps representations of both levels of context and contains routines for using them in handling two discourse-level phenomena that were common in the dialogs we collected. First, the global dialog context is used in the resolution of definite noun phrases; i.e., in identifying those

concepts referred to by the noun phrases in an utterance that are definitely determined (e.g., the nuclear sub). Secondly, the immediate context of an utterance -- syntactic and semantic information from the preceding utterance -- are used in the interpretation of elliptical expressions.; e.g., in expanding a noun phrase (the submarine?) in context (Who owns the carrier?) into a full utterance (Who owns the submarine?).

Encoding global context consists, in essence, of separating out that subset of a system's total knowledge base that is relevant at a given point in a dialog. The goal is to determine and represent what is in the user's focus of attention. For a semantic network knowledge representation, what is required is to change the homogeneous nature of the network by highlighting certain nodes and arcs. In the SRI speech-understanding system this highlighting is achieved by partitioning the semantic network into focus spaces. Chapter IX includes a brief description of this representation and describes its use in resolving definite noun phrases.

The immediately preceding utterance provides the context needed for interpreting an elliptical utterance (or phrase). In the speech understanding system the syntactic and semantic frameworks needed for building an interpretation of an elliptical utterance are provided by attributes recorded in the parse tree of the preceding utterance. Chapter X describes several forms of ellipsis and the mechanisms for handling ellipsis in the system. Of particular interest is the use of

the parse-time semantic network partition in limiting the work done for interpretation of elliptical utterances.

B. DIALOG COLLECTION AND ANALYSIS

'Task-oriented dialogs' are dialogs between two people cooperating to complete some task, where 'task' encompasses real-life activities that are directed toward achieving a particular goal and that can be broken down into small steps, each having its own goal. Examples of tasks include repairing faulty equipment, building a house, carrying out a chemistry experiment, and solving algebra word problems. Task-oriented dialogs occur normally as a master craftsman instructs an apprentice, two mechanics work together to repair a car, and as a teacher guides a student in a chemistry lab. The major characteristics of these dialogs are that both participants are aware of the task to be performed and that communication between the participants is necessary for accomplishing it.

The tasks considered in this research have one further characteristic: they are tasks for which it is feasible to consider a computer taking the role of one of the participants sometime in the not too distant future. In particular, we have investigated situations in which the computer guides a person performing a task. Interest in such dialogs arose in part from considering the language requirements of a computer-based consultant system. A description of initial steps toward building such a system may be found in Hart (1975).

In addition to the task-oriented dialogs, we collected a set of 'question-answering' dialogs. Question-answering dialogs occur when one person asks another (or a computer system) a series of questions in order to help solve some problem. They are distinguished from task dialogs mostly in that the answerer cannot be viewed as sharing a goal in common with the questioner. Although short question-answering dialogs are common in everyday conversation, extended ones (more than five or so questions) are more frequent in communications with computers, for example, in a sequence of queries to a computer data base. In the dialogs that were collected, a person queried a data base in order to solve an assigned problem. Solution required interaction with the data base. To avoid confusion with other kinds of question-answering dialogs, these dialogs will be referred to as data base dialogs in the remainder of the discussion. Data base dialogs differ from task-oriented dialogs both in the degree of structure present and in the influence of the task or problem on that structure. As a result of the differences in amount and kind of problem structure, there are significant differences in the kind of language occurring in the two kinds of dialog.

Task-oriented dialogs are a good source of unbiased data on discourse. Concentration on the performance of a task keeps the participants from becoming self-conscious about their language. The resulting dialogs are spontaneous and unrehearsed. The data base dialogs are somewhat less spontaneous. The less realistic nature of the

assigned problems contributed to the subjects in these dialogs being more self-conscious than those in the task dialogs.

The dialogs described in this report were both written and spoken. To simplify the following discussion, the term 'speaker' will be used to refer to the transmitter of a message and 'hearer' to the receiver even though some of the transmissions were typed.

Section C contains a description of the method of dialog collection. Section D presents an analysis of the dialogs; the major emphasis here will be on the task-oriented dialogs; the data base dialogs will be used to provide contrast. Finally, some other natural language data are examined and future areas of analysis indicated.

C. COLLECTION OF THE DIALOGS

1. PURPOSE

The main purpose of dialog collection was to provide data for determining characteristics of the language used when people communicate for the purpose of solving a problem. Since the goal of the dialog analysis was to determine the language demands a person would make on a computer system, the ideal context for collection would be one in which a person was interacting with a computer. But this is a 'Catch-22' situation: data are needed to guide the design of the system. The best that can be done is to simulate this setup. At the start it was not clear whether the language people use in communicating with one another

would differ from the language they might use in communicating with a computer. We expected that people's ideas of the capabilities of computers would influence the language they used, even if they were told that the system understood English. In the task situation we were able to collect data both of the language used when two people were interacting directly and of the language used when one person thought the other dialog participant was a computer. Our intuition proved correct: the language used in communicating with a 'computer' was different.

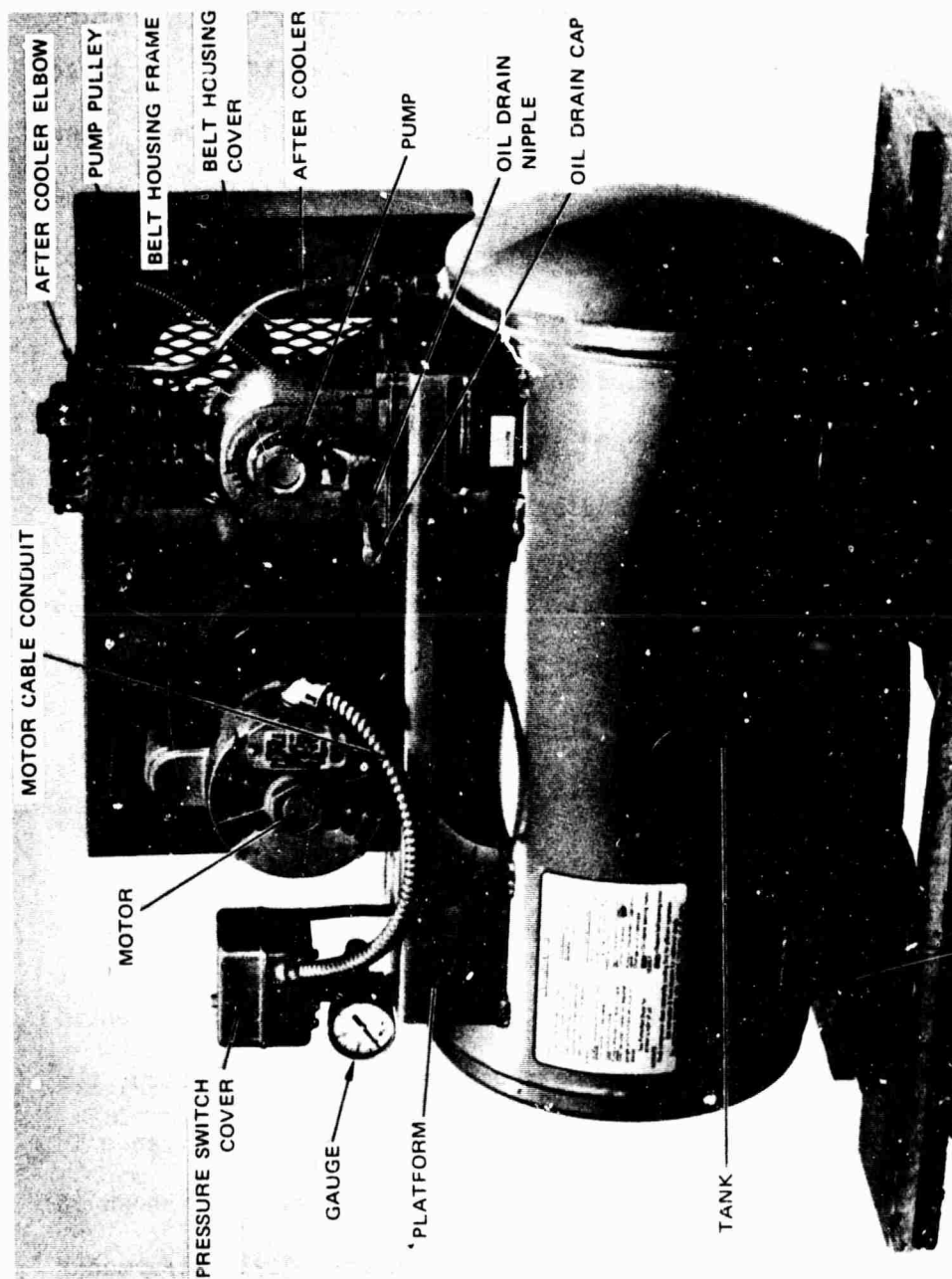
Chapanis (1975) has been interested in characterizing differences in language use across different modes of communication. For example, he investigated differences in measures such as number of sentences, number of words, and number of "nounlike" words across modes such as handwriting, typing, and speaking. In addition, he examined the differences in time required for problem solution across the different modes of communication. His analyses are statistical; they provide information about how the language used in the various modes differs. Although such statistical measures provide some indication of the desirability of one mode over another and of the effect of the mode on the language used, they do not provide the information required for building a computer language-understanding system. For that, information is needed on the particular words used and on how they are put together in utterances to provide meaningful communication.

The analysis reported here is of a different sort: it is concerned with taking a single mode (actually a small number of very similar modes) of communication and characterizing the range of language devices used to achieve successful communication of an idea. A large number of different questions can be asked along these lines. They include sentence-level questions like "How many different sentence structures occur?", "Do some occur more frequently than others?", and "In what context?"; intersentential questions like "What links are there from one utterance to another?"; and more global questions like "Does a dialog have some overall structure?" These questions must be answered before a complete language-understanding system can be built.

The emphasis of the analysis presented here will be on discourse-level phenomena, and in particular, on the structure of the dialogs, the relation between dialog and task, and the kinds of references used for identifying objects.

2. TASK DIALOGS

The main task used for collection of data on task-oriented language was assembly of part of an air compressor. In addition, two dialogs were collected in which an expert plumber provided guidance in the repair of a leaky faucet. A sketch of an aircompressor is shown in Figure VIII-1. For the purposes of understanding the dialog fragments in this report, it is important to note the pump, the pump pulley, the platform, the aftercooler, the belt-housing frame and cover, and the



SA-1530-70R

FIGURE VIII-1 A SMALL AIR COMPRESSOR

connections between these parts. Tasks involving both high-level assembly -- installing the pump and belt -- and lower-level assembly -- putting the pump together -- were used.

The participants in each of the dialogs were an 'expert' (E) and an 'apprentice' (A). The experts, in addition to being skilled at mechanical tasks, were familiar with the compressor and the tools used in assembling and disassembling it. Before participating in the dialogs, the experts performed the task themselves and then had a practice session instructing someone else. None of the apprentices was familiar with the air compressor; in general mechanical knowledge, they ranged from complete novices to amateur auto mechanics.

Dialogs were collected under a variety of conditions. The visual contact between participants was varied to determine the effects of limited vision and to collect data on descriptions. In the first experiments, E and A were allowed to communicate freely; they interrupted each other frequently. For the next set of experiments, the ability to interrupt was removed to see what effect this would have on communication and task accomplishment. Finally, the information given to A about E was varied.

The dialogs fall into four classes:

(a) Free, with vision: E and A were in the same room; they were able to see each other; verbal communication was spoken; no restrictions were placed on language use. The

only instructions were to complete the task. The only restriction was that E could only instruct A; he could not help DO the task. In this setup, then, E could see A, monitor what A was doing, and notice where A put tools and parts. E and A were free to interrupt one another.

(b) Free, with no vision: the conditions were the same as (a) except that E was not able to see what A was doing.

(c) Restricted and aware: both visual and verbal communication were restricted in these protocols. The experimental set-up is shown in Figure VIII-2. Verbal communication passed through a monitor who was responsible for assuring that E and A did not interrupt each other. In these dialogs A spoke, and the monitor typed the message; E typed a response and the monitor read it to A. Computer terminals were used solely so that transcripts could be easily obtained. E was able to get 'still' pictures from the television camera. They had to be requested; normally, the camera was focused on a blank wall. In these experiments, A was informed that the experiment was a simulation of a computer system. Hence, A was aware that E was a person.

(d) Restricted and unaware: the experimental setup was the same as in Condition (c), but A was told that E was a

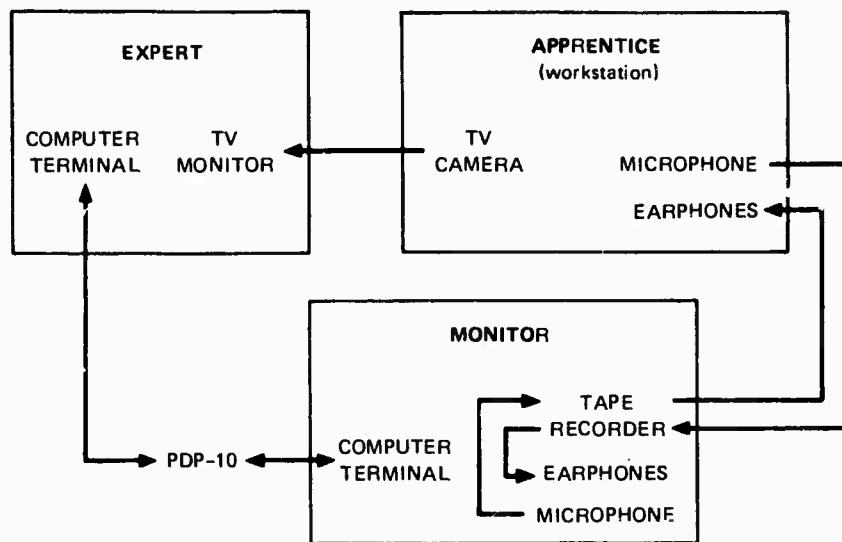


FIGURE VIII-2 EXPERIMENTAL SETUP FOR RESTRICTED DIALOGS

computer system. In each case we determined after the protocol was collected and before explaining the true nature of the experiment that A believed that a computer system was serving as expert.

3. DATA BASE DIALOGS

The data base experiments were designed to collect data on the language people would use if they had verbal access to a data base. In order to collect realistic data, it was necessary to provide people with a specific problem, requiring information from the data base. Again the purpose was to make their language as unself-conscious as possible.

Detailed descriptions of the procedures for collecting the data together with examples are in Deutsch (1974b) and Silva (1975).

The data base used for these dialog experiments contained information about the ships of the United States, British, and Russian fleets. In the first set of dialogs, the subjects were given charts (similar to the ones found in naval reports) to fill out, and two short problems to solve. They were instructed to ask for information from an analyst, who answered using material from the data base. The subjects and analysts were in the same room but were not allowed to interrupt one another or to view each other's materials. For these problems, no additional information can be obtained from the subject and the analyst being able to see one another.

The second set of dialogs used a revised data base containing information on U.S. and Russian ships in the Mediterranean. Subjects were given one long problem to solve for which they needed information in the data base. The subjects were not restricted in their use of language. Their queries were translated into data base queries and typed to a computer data base system by an 'operator'. The answers were read back to the subject.

D. ANALYSIS OF THE DIALOGS

1. OVERVIEW

This section presents analyses of the data gathered. There were marked contrasts between the task dialogs and the data base dialogs in word use, utterance structure, and overall dialog structure. These differences stemmed from the fact that in the task situation, both participants knew and were responsible for the 'solution' of the task, but in the data base dialogs, only the subject was responsible for the problem solution. Furthermore, the task dialogs involved tasks that break down into subtasks. The relationship between subtasks is well-defined. As a result, successive utterances in the task dialogs had strong links. In contrast, the information needed for solution of the data base problems could be asked for in a variety of ways (i.e., a variety of question sequences). There was no necessary dependence of a query on what preceded or followed it.

Ten task dialogs were collected: one under Condition (a), and three each under Conditions (b), (c), and (d). The major difference in language between the free dialogs and the restricted dialogs was the frequent occurrence of interruptions in the free dialogs. Expert and apprentice cooperated on completing utterances as well as on completing the task. The dialog segments in Figure VIII-3 illustrate this cooperative aspect of the interruption. Lines (5)-(6), (9)-(13), and (17)-(18) are the most direct examples. In the first two cases, E is

- (1) E: . . . and those are to be inserted in the side of the motor . . . in the side of the rear of the motor . . .
- (2) A: Uh hm.
- (3) E: . . . and . . .
- (4) A: . . . I see it . . .
- (5) E: O.K. and each wire is to be attached to a . . .
- (6) A: One of those bolt things here?
- (7) E: bolt? . . . yes.
- * * *
- (8) A: . . . now should I unscrew the nuts from the bolts?
- (9) E: No. The wire goes on top of that . . . on top of the nuts that are on there . . .
- (10) A: I see . . .
- (11) E: . . . and there're . . .
- (12) A: Other nuts.
- (13) E: . . . there are other nuts . . .
- * * *
- (14) E: The washer will be the last thing that . . .
- (15) A: The washer will be last . . .
- (16) E: The last item that will be on it.
- (17) A: O.K. Then this little plastic thing
- (18) E: With the holes in it.

Figure VIII-3. FRAGMENTS OF COOPERATIVE DIALOGS

pausing in search of the 'right' phrase when A fills it in. In (17)-(18), E gives a similar kind of aid to A. Lines (2) and (4) are typical of the kind of ongoing mutual support of the two participants. A indicates an understanding of what has been said so far, so E may continue. This support is also evident in the echoing of (14)-(16). The kind of fragment resulting from these interruptions was more than we wanted to attempt to handle in an initial speech understanding system. We surmised that not allowing the participants to interrupt would not seriously hamper problem solution. Chapanis (1973) has evidence that supports this hypothesis. The restricted dialogs were designed to eliminate interruptions. The design of the experiment for restricted dialogs closely resembles Chapanis' setup but was designed independently.

The different visibility conditions had several different effects on the dialog. Robinson (1975a) discusses some of these. The most pronounced difference was in the kind of descriptions that resulted. Figure VIII-4 shows the most blatant contrast found in the dialogs.

If visual information is shared, that common information can be used in descriptions. In the protocols with restricted dialog and limited vision, E often asked for a still picture in order to use this kind of information. The dialog fragment in Figure VIII-5 is an example.

WITH VISION:

E: You have a top piece with a KNURLED section that you can take ahold of.

A: What's a knurled section?

E: You've got your fingers on it.

WITHOUT VISION:

E: Now underneath is what they call a cap assembly. It has a KNURLED face around it.

A: What does knurled mean?

E: Little lines running up and down on it so you can take ahold of it.

Figure VIII-4. DESCRIPTION OF "KNURLED" WITH AND WITHOUT VISION

E: Use the ratchet wrench on the top and hold the nut stationary on the bottom with a box wrench.

A: What is a ratchet wrench?

E: Show me the table.

E: The ratchet wrench is the object lying between the wheel puller and the box wrenches on the table.

Figure VIII-5. USING VISION TO HELP WITH A DESCRIPTION

The difficulty of giving descriptions without the aid of shared visual information is best illustrated by the fragment in Figure VIII-6. A more extensive discussion of the descriptions found in the dialogs and some of their characteristics is presented later in Section D.7.

E: O.K., uh . . . now, we need to attach the um . . . conduit to the motor. . . the conduit is the uh . . . the covering around the wire that you . . . uh . . . were working with earlier. Um, there is a small part um . . . oh brother . . .

A: Now, wait a s . . . the conduit is the cover to the wires?

E: Yes. and . . .

A: Oh, I see, there's a part that . . . a part that's supposed to go over it . . .

E: Yes . .

A: I see . . it looks just the right shape, too. Ah hah! yes . . .

E: Wonderful, since I did not know how to describe the part!

Figure VIII-6. DIFFICULTIES IN EXPLAINING AN UNFAMILIAR
COMPLEX OBJECT

Four of the ten task dialogs form the core data of the analysis: two each of the dialogs occurring under the two restricted language conditions [Conditions (c) and (d)]. These conditions were selected because they were closest to the situations that would occur in any person-computer interaction in the near future. Since each of the dialogs took between 40 minutes and two hours and consisted of between 120 and 250 lines, this constitutes a large body of data.

Most of the attributes discussed below occurred to some extent in all of the dialogs. Interesting phenomena that occurred in isolated dialogs also will be pointed out.

In addition to the ten task-oriented dialogs, five data base dialogs were analyzed. Two dialogs were chosen as representative of the dialogs collected during the first experiment. All three dialogs from the second set were analyzed. Again, although the number of dialogs is small, the amount of data in each dialog is quite large. The dialogs in the first set are over 100 lines long and represent approximately 30 minutes of speaking time. The dialogs from the second set each represent over an hour of dialog. It was necessary to look at long segments of dialog to get the data needed, since the range of discourse phenomena was of interest rather than statistics on what occurs most often.

2. THE STRUCTURE OF THE DIALOGS

a. TASK-ORIENTED DIALOGS

The most interesting characteristic of task-oriented dialogs is that they have a structure that closely parallels the structure of the task being performed. The whole dialog is segmented into subdialogs, which themselves may break down into subdialogs, just as the task breaks down into subtasks, which themselves may be decomposable. For example, the 'task' of making a cake has subtasks of preparing the batter, actually baking the cake, and icing the cake. A recipe (or television cooking program description) contains distinct parts for each of these subtasks. Likewise, the compressor task of installing the pump breaks down into attaching the pump, attaching the

pump pulley, attaching the belt, and several other tasks. Attaching the pump breaks down into positioning the pump and actually attaching it. An analysis of the dialogs for the pump installation task reveals that they fall into subdialogs paralleling these subtasks. The correspondence between task structure and dialog structure plays a crucial role in determining the context in which an utterance is interpreted. It is particularly important for the interpretation of references (see Section D.3).

Several linguistic devices indicate the segmentation of a dialog. As an example, consider the use of "when". The subdialog corresponding to a task ends, or is 'closed', when the task it parallels is completed. If reference needs to be made later to an object or action in that subdialog, the subdialog must be reopened. "When" provides one means of accomplishing this. The utterance, "A little metal semicircle fell off when I took the wheel off" is meant to reinvoke the entire context of taking the wheel off in order to determine the meaning of the metal semicircle falling out.

There are different ways to open and close subdialogs. The most common opening is a statement of the goal of the task. Frequently this is preceded by "next" or "the next step is". If A opens the task, just the plain task description may be used. For example, "I'm tightening the motor mount bolts". Correspondingly, the most frequent kind of task closure is a report of completion of the task. Frequently, this is preceded by an "O.K." Section D.2.d contains more examples.

Another indication of the segmentation phenomenon comes from the use of pronouns to refer back over long portions of discourse. After a subdialog is closed, a pronoun may be used to refer to objects in the higher level task that contains the subtask corresponding to the subdialog. This is the case in the dialog example of Figure VIII-7.

E: Good morning. I would like for you to reassemble the compressor.

. . . .

E: I suggest you begin by attaching the pump to the platform

. . . (other subtasks)

E: Good. All that remains then is to attach the belt housing cover to the belt housing frame.

A: All right. I assume the hole in the housing cover opens to the pump pulley rather than to the motor pulley.

E: Yes that is correct. The pump pulley also acts as a fan to cool the pump.

A: Fine. Thank you.

A: All right the belt housing cover is on and tightened down.

(30 minutes + 60 utterances after beginning)

E: Fine. Now let's see if it works.

Figure VIII-7. PRONOUN USE REFLECTING DIALOG STRUCTURE

The completion of the belt housing cover attachment closes the subtask of installing the cover. The "it" in the last utterance refers to the air compressor last mentioned over a half-hour

before. This use of "it" is not unique. In fact, similar expressions containing "it" references to the air compressor occurred in three of the four core dialogs. There were also several instances of pronoun references skipping over smaller pieces of dialog. In every case, the pieces skipped over were whole segments relating to some distinct subtask or subtasks.

This segmentation is a reflection of an important underlying phenomenon: as different parts of the task are performed, different objects and actions come into 'focus'. The segmentation of dialogs is a reflection of the shifts of focus with time. When a subtask is completed, it fades from focus. However, the higher level (parent) task remains in focus. Hence, when a sibling subtask is performed, the concepts in the parent -- but not those in the completed subtask -- are in focus and affect the use of referring expressions like pronouns. This notion of focus is closely related to Chafe's notion of 'foregrounding' (Chafe, 1972). Both are discussed in more detail in Chapter IX and in Grosz (1977).

b. THE DATA BASE DIALOGS

The data base dialogs did not exhibit the same kind of segmentation, but there was definite evidence of groups of closely related utterances. The amount of segmentation evident in these dialogs differed according to the problem being solved.

The dialogs for the chart-filling-out problems had no global structure although there were sequences of related utterances. The sentence-to-sentence links were most evident from the use of elliptical sentence fragments. The sequence in Figure VIII-8 illustrates one utterance providing context so that only a phrase suffices as a complete utterance in that it conveys a whole question. As Chapter X shows, the use of ellipses is a local discourse feature; it operates only between adjacent utterances.

S: What's the surface displacement of the Lafayette class?

A: 7300 tons.

S: What's the submerged displacement?

A: 8200 tons.

S: The length?

A: 425 feet.

S: Number of torpedo tubes?

Figure VIII-8. A SEQUENCE OF ELLIPTICAL SENTENCE FRAGMENTS

The dialogs for the short problems exhibit a slightly larger grouping of utterances. Some evidence of shifting of focus over subproblems appears. The dialog fragment in Figure VIII-9 is a self-contained unit. The immediately preceding utterance was about British diesel patrol submarines. The utterances following this subdialog were about submarines other than the Yankee and the Hotel II. The subdialog

S: What classes of USSR submarines are there?

A: <answer>

S: How many of tho are nuclear ballistic missile sub-marines?

A: Two.

S: What are they?

A: Yankee, Hotel II.

S: How many tubes does the Yankee have?

A: Eight.

S: *That's torpedo tubes, right?

A: *Eight.

S: And, how many torpedo tubes and missile launchers for the Hotel II?

A: Ten torpedo tubes, three missile launchers.

S: What is the submerged speed for the Yankee and Hotel II?

A: <answer>

Figure VIII-9. A DATA BASE QUERY SUBDIALOG

itself narrows from considering all Soviet submarines to asking about attributes of two particular submarines. There is a short subdialog inside the subdialog itself. The two starred utterances form a clarification-question/answer pair. Although such segments appear in these dialogs, there are not very many of them. This is the longest sequence that appeared; the others were only six to eight utterances

long. Most of the dialog still consisted of sequences of utterances related locally but without structure.

The dialogs for the longer problems exhibit more structuring. Figure VIII-9 gives an example. The questioning moves from determining elements of a particular class of submarines to a subdialog covering attributes of two of those ships. Openings and closings of these subdialogs are less clear than those for the task dialogs. As a result, the segmentation is harder to detect.

What distinguishes the data base dialogs most from the task dialogs is the lack of any intermediate structure. There are local discourse phenomena tying adjacent utterances together, and there is some structure provided by the overall problem, but there is little relating the local segments together into bigger segments. As the problems posed to the subjects get larger, intermediate level organization appears. What seems to happen with these problems is that a solution breaks down into some recognizable substeps and the dialogs fall into segments according to these substeps. There is a continuum, then, of which we have only a few sample points, from the totally unstructured chart-filling dialogs to the highly structured task dialogs.

c. KINDS OF SUBDIALOGS

The subdialogs we have discussed so far are task or problem related; they can be linked directly to some substep of the task being attempted. Several other kinds of subdialogs occur: general question answering, clarification, and communication channel related. Some of these are quite short, only a pair of utterances, but they are all distinguishable as separate from the surrounding dialog and cohesive as a unit.

General question-and-answer subdialogs include subdialogs related to identifying objects in the domain (e.g., "What's a motor bolt?"), describing tool use ["How is this (wheelpuller) used?"], identifying the right tool to be used or seeing if a better tool is available (e.g., the expert asking "What tools are you using?"), making sure no blatant error occurs in performing the task (e.g., the apprentice asking, "Will this require some effort?"), and testing whether a task was performed correctly (e.g., "How tight should the bolts be?"). The data base dialogs contain only a few general question-answering dialogs; they are all concerned with terminology, e.g., "What do you mean by deployment?"

Two kinds of subdialogs fall in-between subtask and general question answering. They are clearly related to the task being done but are also general questions. First, there are questions about why a certain part or step is needed (e.g., "What is the key for?").

Second, there are requests by the apprentice for alternative ways of doing some task (e.g., "Do you have another way to get the nuts in underneath the platform?").

Both the task and the data base dialogs contain pairs of exchanges whose purpose is to determine that the previous message was heard correctly or to have a missed message resent. The middle two lines of the dialog in Figure VIII-10 are an example of this kind of subdialog. Requests for retransmission include statements like "What was that again?" and "Please repeat the last instruction."

A: One of them is at 14 degrees E, 34 degrees N.

S: 34 degrees you said?

A: Yes.

S: O.K.

Figure VIII-10. A SUBDIALOG CHECKING PREVIOUS MESSAGE

There are also subdialogs where one participant wants to make sure that the other participant means the same thing as he does. This kind occurs in the starred sequence of the dialog fragment of Figure VIII-9.

d. OPENING AND CLOSING OF SUBDIALOGS

Task subdialogs may be opened by either expert or apprentice. In the dialogs that were examined, expert openings were always statements of the subtask goal. Sometimes the statement was augmented by a sequencing expression such as "next" or "now". Subdialogs opened by apprentices also included subtask goal statements, but these could be embedded either in statements indicating the task was being, or about to be, performed, or in statements requesting information on how to perform the task. Frequently, a pair of utterances serves to open a subtask. This happens when A asks for the next task, as in the following:

A: What should I do now?

E: Remove the pump.

Alternatively, a pair may result from A asking how to do some task, leading to E giving a subtask specification, as in the pair:

A: How do I remove the pump?

E: First remove the flywheel.

Such pairs occurred both when A knew what task was next but not how to do it and when E gave the task and A needed more specification. As an example, consider the preceding four utterances as part of a single dialog.

Task subdialogs that occurred when the apprentice ran into trouble were opened by a statement of the problem. Similarly, subdialogs for checking task performance were opened by the expert

asking if some goal had been achieved or was in the process of being achieved.

The most typical closings of subdialogs were through statements like "O.K." or ones indicating that a task goal was completed. Often a combination of these was used. These closings are explicit; implicit closings also occurred quite frequently. Typically, A would indicate that a subtask was finished by asking for the next subtask. In these cases, the same statement might serve both to close an old subdialog and to open a new one.

Question-answering subdialogs are always opened by a question about some part, tool, task, or problem. In the protocols collected, some of these subdialogs were closed with a direct answer. In other cases, a long series of exchanges occurred before the answer was arrived at. Only some short sequences contained a closing "O.K." or other explicit indication from A. Almost all of the longer sequences ended with such a communication.

e. MULTIPLE USES OF O.K.

Robinson (1975a) pointed out the use of "O.K." as an acknowledgment that the preceding message has been received. This is only one of four meanings this small word took on in the dialogs. In particular, "O.K." was used at different times to mean:

- * I heard you
- * I heard you and I understand
- * I heard you, I understand, and I am now doing (or will do) what you said
- * I'm finished (O.K. what next?)

Figure VIII-11 contains examples of each of these meanings.

O.K. -- I HEARD YOU:

E: Loosen the motor bolts and slide the motor toward the pump.

A: O.K. What's a motor bolt.

O.K. -- I HEARD YOU AND I UNDERSTAND:

E: That's the center portion of the wheel. Point at where you think it is. Show it to me please.

A: O.K. Just a sec.

O.K. -- I HEARD YOU, I UNDERSTAND, AND I'M DOING WHAT YOU SAID:

E: First loosen the two allen head setscrews holding it to the shaft, then pull it off.

A: O.K.

A: I can only find one setscrew. Where's the other one.

O.K. -- I'M FINISHED:

A: O.K. All the bolts are off.

Figure VIII-11. DIFFERENT USES OF "O.K."

Each of these uses of "O.K." requires a different response from the hearer. Often the indication of which one is meant comes from the next statement in the dialog. Although the time between

the preceding statement and the "O.K." is often a clue to which meaning is intended, it is not always a reliable indication.

The main problem for building a computer system comes from distinguishing the first three levels of "O.K." from the fourth. In the task domain, level 2 never occurred where level 3 was applicable (though one can imagine it in some situations, like a child being told to make his bed). The distinction between level 1 and levels 2 and 3 will be immediately evident from the utterance following the "O.K." Furthermore, no ambiguity problems can arise from this distinction since it does not have any impact on change of focus. Level 4, on the other hand, does indicate a change of focus: 2 once a task is completed, focus shifts to a new task. At present, the best strategy for interpreting "O.K." seems to be to wait for the next utterance to determine if a shift of focus is intended.

Figure VIII-12 contains a dialog fragment illustrating one of the problems that arise from the use of "O.K." for closing a subdialog. In line (4), A indicates completion of part of the 'open-valve' task. In line (5), E gives the next task; he has closed the whole 'open-valve' task. However, from line (6) it is clear that A thinks another subtask may be involved in the 'open-valve task'. To answer (6), E must re-open the closed (for him) 'open-valve' task and its corresponding subdialog.

- (1) E: Open the top of the valve and let the water out. Just open the faucet up on top. Just like you were going to turn the water on.
- (2) A: Oh, like I'm going to turn the water on. O.K.
- (3) E: Now, that'll relieve the pressure.
- (4) A: O.K. some water came out.
- (5) E: Now the next thing you do, you take an allen wrench . . .
- (6) A: Do I leave it on or turn it back off?
- (7) E: It doesn't make any difference.
- (8) A: O.K.

Figure VIII-12. A MISUNDERSTOOD "O.K."

f. MULTIPLE OPEN SUBTASKS

The preceding discussion has centered around the idea of only one task being under discussion at any time and hence providing focus for the dialog. However, some examples of more than one focus were encountered in the dialogs analyzed. These fell into two categories: 'hypothetical' and 'competition'. In the hypothetical case, one task was being performed, but a future one was being considered. Although the task being performed was a lengthy one, there were no problems, so the apprentice asked about how to perform some future task, or what would happen if some task were performed differently. In all such instances, both A and E seemed comfortable with the multiple foci.

In the competition case, however, E and A appeared to be competing for who would determine what would get discussed. Although both could handle the dual foci, at least one of the two always seemed annoyed. The annoyance was manifest both through repetition of statements and from the tone of message communicated orally. In all cases, the maintenance of multiple foci did not last more than two or three exchanges.

3. REFERENCE

The importance of the link between task structure and dialog structure and the need for representing focus of attention are most clearly seen when examining the use of 'referring expressions'. The utterances in a dialog (or any discourse) comprise two kinds of information. Some of the information has been introduced previously into the discourse; in previous research, such information has been labeled 'given' or 'old'. Other information is 'new'; it is being introduced into the discourse by this utterance. Understanding an utterance requires identifying the given concepts in memory and attaching the new information to them. The term 'referring expression' denotes those parts of an utterance that communicate given information.

Determining the information and processes needed to identify the object meant by a referring expression, resolving, that is, a reference, was a primary goal of the dialog analysis. Because definite noun phrases are the most common form of referring expression, the

analysis focused on the use of these phrases. For some of the analysis it will be useful to distinguish two kinds of definite noun phrases: pronouns and nonpronominal definite noun phrases. In the following discussion, the term 'DEFNP' will be used to refer to nonpronominal definite noun phrases only. The basis of this distinction arises from the different processes needed for resolving pronoun references and DEFNPs.

Resolution of DEFNPs is basically a retrieval process. The context in which an utterance appears -- both the surrounding non-linguistic environment and the global linguistic context of the preceding discourse -- is crucial to the resolution process for the DEFNPs in the utterance. The immediate linguistic context and, especially, the sentential context of the referent itself, are not important. For most pronouns, the opposite is true. Unlike DEFNPs, pronouns carry almost no information themselves. The immediate linguistic context of the preceding utterance (and preceding clauses in the same utterance) supplies candidates for the referents; sentential context provides restrictions for choosing among them. Global context is not very important. The exception for pronouns is the previously mentioned use of pronouns to refer back over long portions of dialog. In these instances, the global context supplies candidates. The process is basically one of retrieval. However, the lack of semantic information in the pronoun makes sentential context necessary for choosing among the candidates. This use of pronouns is similar to the

'pragmatic anaphora' in Harkamer and Sag (1976). In essence, resolution of these pronouns, like DEFNPs, is basically a global semantic process. Resolution of other pronouns is more local and more syntactic.

There are several ways in which the object referred to by a DEFNP may be evident in the discourse context. The simplest case is when the object was explicitly mentioned in a preceding utterance. Chafe (1972) pointed out another use of DEFNPs: to refer to objects that are 'foregrounded'. These are objects that are not explicitly mentioned in the discourse but are so closely coupled to some object which has been that they may be considered "in the consciousness of the hearer" (Chafe, 1974) and, hence, may be referred to definitely. For example, in the sequence,

E: Are you using the socket wrench?

A: Yes. The socket fell off ...

"the socket" has not been previously mentioned but is foregrounded when "the socket wrench" is identified.

A problem of particular interest in resolving references is determining where to search for referents: how far back in the dialog is it necessary to go? Searching the whole preceding discourse may be quite time consuming. The necessity of considering foregrounded concepts as well as those explicitly mentioned makes searching the whole dialog unreasonable.

Chafe (1972) noted that the time (or, its analog, distance, in a text) between utterances affected whether or not a definite reference could be used. He also remarked that it was not clear how much discourse could occur before an object ceased to be foregrounded. Most language understanding systems use some time measure as the sole basis for considering objects as referents of definite noun phrases. The system of Norman et al. (1975) has a concept of 'working memory', but objects must be explicitly rementioned in order to stay in this memory. The examples presented in Section D.2 illustrate that time alone is not a sufficient determiner. Whole segments of dialog may be skipped over, and objects not mentioned for a long time may be referred to by definite noun phrases (even pronouns!).

Examination of the references occurring in the task dialogs showed that references operate inside of subdialogs. That is, as long as a subdialog is open, objects introduced into it are referred to by definite noun phrases. We consider these objects 'in focus'. When a subdialog is closed, the objects inside it leave focus and require different references (unless the whole subdialog is reopened or they are first reintroduced in some other subdialog). When a subtask is completed, the definite noun phrases may refer to objects in higher level tasks. For illustrative purposes, consider the simple tree task structure of Figure VIII-13. When task T6 is completed, there is a return to the context of T2 and possibly directly to T1, but there can be no references to objects only in T4 or T5. Objects in T4-T6 cannot

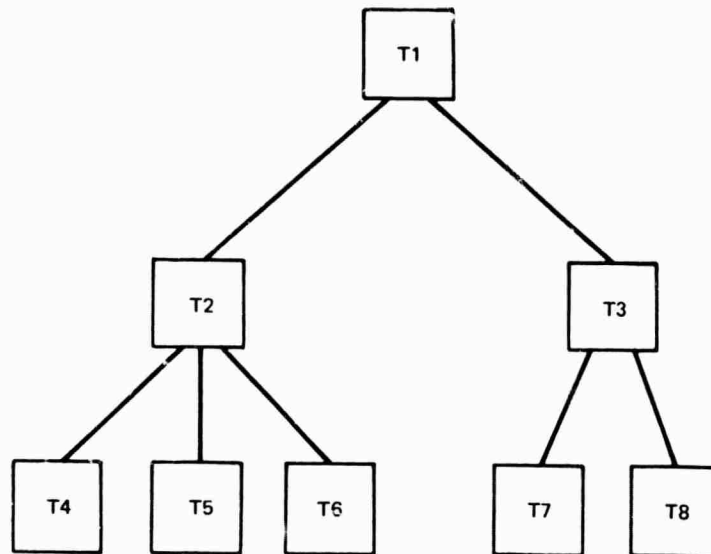


FIGURE VIII-13 A SIMPLE TASK MODEL FOR ILLUSTRATING DIALOG "POPS"

be directly referenced from T7 or T8. When T8 is completed, there may be a 'pop' up to T3 or T1.

Most references can be resolved in terms of the preceding utterances in the subdialog, but this is not of itself sufficient for establishing the existence of segmentation mentioned previously. Those utterances are also the most recent ones. A simpler explanation of the reference retrieval process can be made in terms of the referent being closest in time. If we consider the references that occur after a subdialog has been closed, we can see a place where the subdialog explanation is more powerful than this 'closest in time' explanation. When a subdialog is closed and focus shifts back to a higher level task,

the objects in that higher task get referred to definitely even though they have not been mentioned recently. The use of DEFNPs in this way might be expected, but the use of pronouns for objects not recently mentioned is certainly striking. The example in Section D.2.a is hard to account for if task and dialog structure are ignored.

A second indication of structure comes from the use of plural DEFNPs. Consider again the task structure of Figure VIII-13 and suppose that some bolts B1 are involved in task T2 and another set B2 in task T3. Then, even if some utterance in the end of the subdialog for T2 contains the phrase "the bolts", any reference to "the bolts" once T2 is closed and T3 opened will be taken to mean the set B2. This is true with a combination of singular and plurals also. So if T2 involves a single bolt B, the phrase "the bolts" inside of T3 will not be taken to include B.

In this connection, it is important to point out that people are sensitive to the distinction between singulars and plurals. In the subdialog of Figure VIII-14, E indicates the ambiguity of the phrase "the allen screw" by pointing out the fact that there are two (in addition he indicates that they both need to be tightened).

This subdialog may be contrasted with the one of Figure VIII-15. Here even though the two screws have been mentioned within one exchange of the wheelpuller screw, the phrase "the screw" is totally unambiguous. Completion of the tightening task has closed one subdialog and removed those two screws from focus.

E: Check the alignment of the two pulleys before you tighten the setscrews.

A: Yes. I'm doing that now.

E: O.K.

A: Tightening the allen screw now.

E: O.K. Thank you.

A: That's finished.

E: By the way, there are two setscrews.

Figure VIII-14. SINGULAR/PLURAL DISTINCTIONS

A: How do I remove the flywheel?

E: First loosen the two allen head setscrews holding it to the shaft, then pull it off.

A: The two screws are loose but I'm having trouble getting the wheel off.

E: Use the wheel puller. Do you know how to use it?

A: No.

E: Loosen the screw in the center and place the jaws around the hub of the wheel, then tighten the screw . . .

Figure VIII-15. EFFECT OF SHIFT IN SUBDIALOG ON DEFNPS

Another indication of dialog structure and segmentation comes from considering a dialog with groups of lines removed. If a whole subdialog is removed, the dialog remains coherent. Although it is

sometimes possible to delete some utterances that are not whole subdialogs without damaging coherency, such removals often result in dialog fragments that do not make sense. Removing a question and its answer may not affect coherency. Removing a subdialog opening or closing does.

The one point where definite references in one subdialog get resolved in terms of objects in a closed subdialog is at the crossover point, the set of utterances that provide the transition from one subdialog to the next. The use of "them" in the sequence

A: I've got all four bolts in place.

E: Good. Now tighten them up.

is not only acceptable, but practically necessary. "The bolts" does as well, but "the (four) pump mounting bolts" is confusing; it seems to indicate another set of bolts. This confusion is present even though A's statement with E's "good" ends one subtask (closing the corresponding subdialog) and the remainder of E's statement opens a new subdialog. The objects in the just closed subtask are still in focus through the transition to the new subtask because the two tasks are contiguous in time. Hence, at such transition points 'closeness in time' provides focus.

In a structured discourse, both time and structure need to be taken into account in resolving references. Previous systems have been able to rely on time as the sole basis of definite noun phrase resolution because they have been concerned with unstructured tasks. In

Winograd's (1971) blocks task any instruction can be followed by any other. Although there is utterance-to-utterance cohesion, there is no global cohesion (other than everything being about blocks). This is exactly what happens in the data base domain, too. Norman et al. (1975) report that the time-based algorithm used in their system works on most references in texts. But textual material is edited according to a set of rules that emphasizes the time aspect of reference. To that extent, texts are atypical of the kinds of language people use in direct communication. The dialogs are perfectly comprehensible when being read; it is clear that segmentation is usable in processing text as well as in interactive forms of communication.

4. KINDS OF UTTERANCES

There are marked differences in the kinds of utterances occurring in the task dialogs and in the data base dialogs. Syntactic differences include such things as differences in the number and kinds of WH-questions and differences in the ratios of questions, imperatives, and declaratives. Several of these are enumerated in Section IV, The Language Definition, in Walker et al. (1975). There were also distinct differences in the kinds of utterances in these two sets of dialogs. These differences are manifest on two levels: 'utterance purpose', the overall reason for the utterance (e.g., to convey task information); and 'utterance type' -- the form in which the utterance conveys information (e.g., a request or a response).

Almost all of the utterances in the data base dialogs are questions whose purpose is to get information out of the data base (that being the nature of data base query). In the task domain, there was a wider variety of utterance purposes and also of utterance types. Utterances served three purposes. The majority were 'task related'; they involved such things as describing task steps, identifying parts and tools, and describing progress on a task. Secondly, utterances served as 'sensory substitutes'; these included requests from E, such as "Show me ...", and statements by A, such as "I'm pointing at ...". Finally, some utterances served to establish that the communication channel was still open, for example, the question "Can you hear me?" In addition, several of the "O.K.s" served as channel checkers as well as providing task information.

There were five types of utterances. Most of the utterances were 'requests' for information or 'responses' to such requests. These types include questions about task steps, which tool to use, and how a task step was progressing, and the answers to such questions. Often, however, information was offered without such requests. Some apprentice utterances were 'reports' of progress. These are quite similar to answers to requests like "What are you doing now?" but differ in that they also indicate A's need to communicate his progress. Similarly, E 'imperatives' are quite similar to answers to the question "What should I do next?" but convey E's feeling of task progress rather than A's. Both reports and imperatives are often followed by utterances that serve

merely to 'acknowledge' that a message has been received. "O.K." and "Yes" often function in this way.

Each type of utterance may be followed only by a subset of the other types. Imperatives and reports may be followed by either acknowledgments or combinations of an acknowledgment and a request. In the latter case, if the request immediately follows the imperative or report, the acknowledgment is implicit and may be omitted. Typical requests following imperatives involve questions about parts of the task; typical requests following reports involve checking that some subtask has been done correctly. Reports may also be followed by imperatives. Again, the acknowledgment is implicit.

With one exception, requests and responses come in pairs. In the usual case, requests must be followed by a response. The response may be followed by anything other than another response. The exception occurs with embeddings of questions and answers as in the dialog of Figure VIII-16. In this case a request is followed by another request. Correspondingly, the response is followed by another response. Finally, acknowledgments may be followed by imperatives, requests, or reports. In a sense, an acknowledgment signals that the acknowledging person is ready to receive another message.

Figure VIII-17 contains a segment of dialog containing the five types of task utterances. In this example, each of the imperatives and reports is followed by an acknowledgment. In several cases, the

A: Should I put the bolt on next?
E: Are the setscrews tight?
A: Yes.
E: (OK)(Then) you can put on the belt.

Figure VIII-16. EMBEDDINGS OF REQUESTS AND RESPONSES

acknowledgment is immediately followed by a request. In these cases, the acknowledgment itself is optional. There are examples, in other places, of similar imperatives being followed by requests for information. A similar situation holds for reports; although in this fragment all reports are followed only by acknowledgments, it is also possible to follow them with requests or with a combination of acknowledgment and request.

The utterances in the dialogs vary somewhat along another dimension, that might be called 'response influence': the amount of influence an utterance has on the form and content of the utterance that follows. It is difficult to point at all of the factors influencing this dimension and many utterances are neutral with respect to it, but others are clearly marked. Consider the two sets of utterances in Figure VIII-18. Utterance A1 is neutral with respect to influence. Either party could take over the dialog at this point; neither the form nor the content of the next utterance is indicated. Utterance B1, on the other hand, puts responsibility for the form of the following

E: The pump pulley should be next.
 IMPERATIVE (this direction follows a report indicating completion of the preceding task)

A: Yes uh does the side of the pump pulley with the holes face away from the pump or towards it?
 ACKNOWLEDGMENT FOLLOWED BY A REQUEST FOR INFORMATION

E: Away from the pump.
 RESPONSE

A: All right.
 ACKNOWLEDGMENT

E: Did you insert the key, i.e., the half-moon shaped piece?
 REQUEST

A: Yes I did.
 RESPONSE

E: Be sure and check the alignment of the two pulleys before you tighten the setscrews.
 IMPERATIVE

A: Yes I'm just now fiddling with that.
 ACKNOWLEDGMENT FOLLOWED BY A REPORT

E: O.K.
 ACKNOWLEDGMENT

A: Tightening the allen screw now.
 REPORT

E: O.K. Thank you.
 ACKNOWLEDGMENT

A: That's finished.
 REPORT

Figure VIII-17. UTTERANCE TYPES IN A SAMPLE DIALOG FRAGMENT

- Set A: 1. A: I've finished installing the strap.
2. E: The pump pulley should be next.
3. A: Yes. Does the side of the pump pulley with the holes face away from the pump or towards it?
- Set B: 1. A: Now what should I do?
2. E: Install the pulley on the shaft.
3. A: What is the first thing to do in installing the pulley?

Figure VIII-18. TWO SIMILAR DIALOG FRAGMENTS FOR COMPARING RESPONSE INFLUENCE

utterance on E. Both utterances A2 and B2 are neutral; they are quite similar in what they convey. The responses to them are quite different, though. Utterance A3 exhibits strong influence over the response to it. One of the two alternatives must be picked or some explanation of why neither was given. The preferred response is a simple phrase choosing one of the two options. Utterance B3 is harder to classify. It does not seem entirely neutral since it indicates no choice or narrowing of alternatives by A, but it is not as clearly an abdication as is B1. Imperatives and yes/no questions exhibit strong influence over the form of responses to them.

Subjective evaluation of the dialogs indicates the lack of response-influencing utterances from As who were unsure of the task, and a higher presence in the dialogs with experienced As. Before this kind

of information can get used in a language understanding system, more analysis is needed both on how the information is conveyed and how it is used. One clear use, though, is to indicate familiarity or lack of familiarity with a problem.

5. ELLIPSIS

Elliptical sentence fragments are phrases that function in context as full sentences, although they are only parts of what would constitute a complete sentence. The use of fragments in the task dialogs was quite different from that in the data base dialogs. In the data base dialogs, the fragments all formed part of a series of questions. In each case, the meaning of the fragment could be obtained by finding a similar phrase in the preceding question and substituting the new phrase for the old. An algorithm for handling this kind of fragment is presented in Chapter X. In the task dialogs, fragments occurred as responses to previous requests for information and as qualifying phrases on immediately preceding utterances. As a result, the fragments in the task dialog were patterned on and needed to be interpreted in terms of the immediately preceding utterance.

The most common form of fragment used in response to a request was the one that fit into the WH phrase of the preceding question. This occurs, for example, in

E: What tools are you using?

A: My fingers.

A's response "my fingers" matches the phrase "what tools". Arriving at a complete utterance requires a set of standard syntactic transformations like changing the "you" to an "I". Robinson (1975a) contains a description of the transformations required to interpret this kind of fragment. Secondly, a fragment may occur in response to a choice question. This is the case in the pair

E: Does the side of the pump pulley with the holes face
away from the pump or towards it?

A: Away from the pump.

(In a sense, this is a restricted form of a WH-question. The WH-phrase is replaced by a choice phrase. This could be phrased as a "Which way ..." question).

The use of a fragment to qualify a preceding utterance is illustrated by the sequence

E: Place the key in the slot.

A: Flat side upward?

In each of these cases, the full sentence needed to get an interpretation of the fragment can be derived from transformations on the preceding utterance. When fragments appear as answers to questions (the first two examples), the questions themselves provide an indication of where the fragment fits in. In the last example, this is not the case. There is no place marked by a WH-phrase to indicate a slot for the fragment. Instead, the fragment fills an optional slot in the sentence structure (for verb complements), which was not used in the first utterance of the pair.

6. LEXICON

Analysis of the words occurring in the dialogs is necessary to determine both the size of lexicon and the breadth of concepts present. Section IV, The Language Definition, in Walker et al. (1975) contains a description of the kinds of words found in the data base dialogs. In this section, only the task-oriented dialogs will be considered. In the following analysis, different forms of the same root were not distinguished. For example, "bolt", "bolted", and "bolts" were treated as identical.

One of the most interesting results was that only 520 different words occurred in the four core dialogs. (There were approximately 8000 words in the dialogs -- not including occurrences of the articles "a" and "the"). Malhotra's (1975) results confirm our finding that only a small number of words seem to be required for communication in a limited domain.

Of the 520 words occurring in the four core dialogs, only 100 are used more than ten times. Although this suggests that most of the communication is achieved by a small core lexicon, it is important to realize that many words occurring only once or twice are crucial to conveying events that occur and objects that are used only a few times. Half of the words are unique to a particular dialog. However, many of these words are just differences in expressing similar concepts. However, 90 words occur in all four of the dialogs. Of these 90, 74 are

among the 100 words used over ten times. The list appears in Figure VIII-19. The starred words were used fewer than ten times. Since the number of different words in each dialog ranged from 236 to 303, approximately one-third of the words in each dialog occurred in each of the other three dialogs as well. If the dialogs are separated into pairs according to task, then the pairs in each grouping share over half of their words (142 and 154). These results suggest both a large overlap in concepts, and a large variety in how concepts are expressed.

a	*again	all	allen	*also	and	at	back
be	belt	bolt	box	by	cai.	do	*easy
*enough	*fit	from	get	go	good	*hand	*hard
have	hold	how	I	if	in	it	just
key	know	*like	*long	loose	*more	motor	no
not	now	of	off	ok	on	one	or
out	*over	place	plate	pleasc	pulley	pump	put
screw	see	*seem	should	show	slide	so	*some
tank	that	the	then	there	they	tight	to
top	*toward	turn	two	up	use	way	we
what	*when	where	which	will	with	*work	wrench
you	yes						

Figure VIII-19. WORDS OCCURRING IN ALL FOUR DIALOGS

The two 'naive apprentice' dialogs share 60% of their words. Correspondingly, only 20% of the words in each of the naive apprentice dialogs are unique to that dialog. The other two dialogs each had approximately 30% unique words.

It is dangerous to generalize from such a limited sample; speaker idiosyncrasies cannot be filtered out. However, there are some clear trends, giving indications for system building and suggestions for

future studies. Approximately 140 of the words in the dialogs were task-dependent words; as the task shifts, the need for these words changes. The overlap between the two naive apprentice dialogs suggests that words applicable to low level task descriptions (e.g., specific simple tools, like screwdrivers) get used more often in these dialogs.

If we add a fifth dialog to the analysis that covered a different task but also used an inexperienced apprentice, similar results occur. The number of words increases to 580. Again, over half of the words are unique to some particular dialog. Only 61 words are shared by all of the dialogs. These words, grouped by category, appear in Figure VIII-20. If we consider the three naive apprentice dialogs, the number of shared words is 88. Twenty-six of these words, listed in Figure VIII-21, are missing from at least one of the experienced apprentice dialogs. The number of words shared by the naive apprentice dialogs is less than the number shared by the four 'task-in-common' dialogs, but many of the additions are clearly from more detailed advice being given (e.g., "screwdriver", "align", and "tool").

Although the overlap of words is interesting, it is important not to ignore the large number of words that are unique to some one of the dialogs. The overlap means that, for a given task, a relatively small number of words (significantly less than 1000!) will suffice to cover almost all of what almost every speaker says. The 'unique words' indicate that although many of the concepts being expressed by the performers of the task are the same, there is a wide variability in just

AUXILIARY AND PRO-VERBS

be can do have should will

DOMAIN-RELATED WORDS

bolt box fit go hold place
plate pump put show tight top
turn use

FUNCTION WORDS

a also and by from how
if in no not now of
on out over so that the
then there to up what when
which with

MISCELLANEOUS

good just like ok one please
see two way yes

PRONOUNS

it they

SPEAKER/HEARER IDENTIFIERS

I we you

Figure VIII-20. WORDS OCCURRING IN ALL FIVE DIALOGS,
GROUPED BY CATEGORY

align	around	both	bottom	but
down	end	face	first	groove
hammer	metal	onto	other	remove
right	round	screwdriver	shaft	side
slot	sure	take	thing	took
wheelpuller				

Figure VIII-21. WORDS IN ALL NAIVE APPRENTICE DIALOGS BUT
MISSING IN AT LEAST ONE OF THE OTHERS

how to express those concepts. Analysis at the lexical level is important, but it must be used in conjunction with higher-level syntactic, semantic, and discourse analyses.

7. DESCRIPTIONS

The section on references (Section D.3) concentrated on the identification of objects from the point of view of context, pointing out how such context shifts with task and with time. The linguistic description of an object must distinguish it from all others in the context of speaker and hearer in order for any communication to be possible. The problem of identifying an object mentioned in an utterance on the basis of its description and the problem of generating reasonable descriptions to guide the user are of equal importance in a language understanding system. For this reason, the descriptions in all of the dialogs were examined in an initial attempt at characterizing descriptions.

a. SPECIFICATION

Olson (1970) has shown that the description of an object changes depending on the surrounding objects from which it must be distinguished. So, for example, the same flat round white object was described as "the round one" when a flat square object of similar size and material was present, but as "the white one" when a similarly shaped but black object was present. However, it is clear from the task dialogs and from other data (Freedle, 1972) that description of an object seldom contains only the minimal amount of information necessary to distinguish the object. Descriptions, like the rest of language, are redundant. (Olson, p.266, comments on this phenomenon and the need for further investigation of it.)

What appears to be the case is that the speaker describes an object not in the minimum number of 'bits' of information, but rather in a manner that will enable the hearer to locate the object meant as quickly as possible. Clear distinguishing features (e.g., color, size, and shape) are part of a description precisely because they enable eliminating large numbers of objects as wrong and hence help the hearer to isolate the correct object more quickly.

The use of redundant information (and not just distinguishing information) to speed up the search for a referent can be easily seen from an example. If A asks "What tool should I use?", the response, "The red-handled one." is not satisfactory even if there is only one red-handled tool in the workstation. Processing such a description requires considering too many alternatives. Although A might eventually find the tool, he would certainly question E's choice of description. "The red-handled screwdriver" is more helpful, because it limits the search to screwdrivers. Olson's descriptions were probably as minimal as they were because of the bare environment in which the distinguishing had to be done. In giving a description that minimizes search time, a balance must be reached. Too much information is as harmful as too little. All parts of the description must be processed to make sure the object is the correct one. Furthermore, the hearer may wonder whether he is mistaken, if he thinks he has found the object but there is more description coming. Rather than minimize either just the communication time (including processing of the

description) or just the search time, the combination of communication time and search time must be minimized.

Because the goal of most descriptions in the task dialogs was to enable the hearer to locate an object, the descriptions in the task dialogs were, to some extent, 'procedural'. Either implicitly or explicitly, they described how to locate an object, rather than what the object was in general. For example, the response to "What's a nutdriver?" was "It looks like a screwdriver and is in the yellow case by the wall", rather than the (nonprocedural) definition description, "A tool with a handle on one end and the end shaped to fit over a nut, used for tightening and loosening nuts." This combination of description of the object itself coupled with locational information was quite common in response to questions (e.g., "What's an x?"). In a sense, the speaker was saying, "Keep these properties in mind and look at place Y." It is interesting that the descriptions of the object itself preceded the locational information more often than following it. The location provides a narrowing of focus. What is not clear is why this narrowing occurs after and not before the object properties are given. Possibly, even though narrowing of focus is useful for identification, the question "What is an x?" demands some description of an object's properties first.

b. CATEGORIES OF FEATURES

The features used in the descriptions of objects in the dialogs fell into four categories: physical characteristics, location, analogies, and function. A class name of the object always appeared in initial introductions, but it is not included in this list. Otherwise, the list contains items used in initial introductions as well as in response to questions concerning object identification.

The physical characteristics of the object itself included color, shape (often including the word "shape" as in "the little half-moon shaped part"), size (either absolute or relative), and material of which the object is composed (e.g., "metal").

Location, both physical and in time, of the object were often used. Physical location was specified in response to a "What's a" question. Time references occurred when an object description was embedded in some higher-level statement. For example, "Use the two screws you mentioned earlier", "... the cover to the wires you were working with earlier".

Analogy provides a lot of information in a small package. It occurred most often when any other description would have been long and involved. In addition to the above screwdriver example, there was "it looks like a pocketknife", "it looks like ears sticking out", or "it looks like a y".

Closely related to analogy is the use of "function" to describe an object. Functional descriptions also enable bypassing other more complex descriptions (e.g., of shape). The combination of analogy and functional description often occurs with the phrase "it looks like it does x" (and, in fact it does do x!). Functional descriptions implicitly convey this concept of "looks like" even when it is explicitly stated.

Finally, there is a set of miscellaneous distinguishing features that are best characterized as the absence of something usual or the presence of something atypical. For example, "[you can tell where it goes] by where there is no paint", or "the side with writing on it".

c. PERSPECTIVE

In order for a description to work, it is crucial that it take into account the hearer's point of view. The role of the hearer's physical location is well established. The well-known "Empire State Building" question (you give a different answer to the question "Where is it" to a person in Moscow and a person in New York City) is meant to illustrate this point. In the task domain, words like "left" and "front" must take into account both canonical orientations (the front of a car is the same no matter where you stand relative to it) and hearer orientation.

There is also a nonlocational aspect of the hearer's orientation. Descriptions must be given to a level of detail pertinent for the hearer's skill level. Concepts unfamiliar to the hearer may be introduced, but they must be explained in terms familiar to him. Evidence of such sensitivity to user skill in the dialogs came both from the level of detail of task described and from the description of parts and tools.

8. MISCELLANEOUS OBSERVATIONS

There were several areas in which only limited data are available from the dialogs but which are important for understanding the choices made in generating an utterance and the information conveyed by an utterance. There were clear indications of the influence of one speaker on another, deficiencies in formality, and influence of apprentice skill level.

One question of importance in constructing natural language-understanding systems is the influence of the way the system states things on the language with which it has to deal. Since only two different experts were used in the task dialogs, only one of whom worked with more than two As, it is hard to conclude much from the dialogs. Still there are indications of A's adapting E's language. Adoption of common names is the most common example. "The half-moon shaped piece" gets referred to as "the (woodruff) key" once the name is introduced by E. Similarly "the screws holding the pulley on" become "the (allen

head) setscrews". The transference may be from A to E as well. In one dialog with an experienced A, E adopted terms (such as "pressure register") used by A.

One of the confounding factors in determining language influences is that in the case of two of the dialogs, A thought that E was a computer. In both, the language is more 'formal' than in the other dialogs. In the one that is the most formal, E responded more formally. It is not clear in this case how much of the difference is due to E's speech and how much to A's image of what a computer expert could understand. Although there are clear differences between the 'computer-expert' dialogs and the others, it is hard to point at exactly what aspect of an utterance makes it seem more formal. For example, the utterance,

"Is it correct that the strap is attached to the pump by one of the cylinder head bolts?"

seems more formal than a question that starts simply, "Is the strap ...". Similarly, "I've finished attaching the tubing to the elbow." is less formal than "The elbow and tubing installation is completed." Unfortunately, there are too few data here to decide what is speaker-idiosyncratic and what comes from anticipated computer capabilities. Still, there are enough indications of differences when a computer is thought to be a participant in the dialog that this is an important area for study. Furthermore, that although the As thought they were being helpful by being more formal, in fact the resulting sentences often were more complex.

E's instructions to As varied according to the skill level of A. In almost all cases, E did not know how skilled A was to start with. Although the initial instructions to all As were quite similar, instructions at the end varied substantially. Not only is the amount of detail presented different but also the way in which instructions are given. Dialogs with inexperienced apprentices contain more requests and fewer spontaneous reports. In the dialogs with more experienced apprentices, there are more imperatives to check that steps have been done and fewer giving directions. The clearest example of E moderating his interactions as he determines the skill level of A is in a dialog with an experienced A. Up to a particular point in the dialog, most of E's utterances are directions or answers to requests. Then E starts to give a direction and changes his 'tone'. E types

"OK. Tig XXX OK. Make sure ... are tight."

(The XXX indicates an erasure to the monitor). The important question for builders of computer systems is what information the human expert is using to base his impressions of skill level on. There are clearly several factors involved. A comparison of the few dialogs we have indicates that A's terminology, the level of detail of instruction A asks for, and A's own indication of skill level contribute. More data need to be collected and examined to determine how skill impressions are transmitted and generalized.

Finally, there were a few examples in the dialogs of the kinds of ambiguity that people are and are not willing to tolerate. For

example, the phrase "allen bolts" in the context of attaching the pump pulley was accepted as meaning "allen head screws". Quite often the use of "nut" and "bolt" interchangeably was accepted, but in the dialog of Figure VIII-22 the misuse of "bolt" is not acceptable since it causes confusion about which task is being done.

A: Should I unscrew at the top of the airhose or at the bottom and which of the bolts at the bottom?

(by bolts, A means nuts)

E: Loosen the pipe at the tank (bottom) end and unscrew it completely at the top end.

A: End of what, the pipe or the bolts?

("bolts", really nuts)

E: We're working on the pipe now. Don't worry about the bolts yet.

Figure VIII-22. BOLT/NUT CONFUSION

It is clear that the dialog analyses reported here are really just a beginning. There are many dimensions along which much further analysis must be done. As stated in the introduction to this chapter, the purpose of this part of the research was to determine the scope of discourse phenomena in dialogs with computers, and to provide a basis for initial attempts to incorporate discourse capabilities in a language understanding system.

IX RESOLVING DEFINITE NOUN PHRASES

Prepared by Barbara J. Grosz

CONTENTS:

- A. Introduction
 - 1. Sentential and Dialog Context
 - 2. The Inference Problem
- B. The Focus Space Encoding of Context
 - 1. Extending the Notion of Partitioning
 - 2. Matching in Focus
- C. DEFNP Resolution in Context
 - 1. From Semantics to Discourse
 - 2. Interpreting Complete NPS
 - a. Singular NPS
 - b. Plural NPS
 - c. Modified NPS
 - d. Genitives
 - e. Quantified DEFNPs
 - 3. Augmenting Focus

A. INTRODUCTION

The presence of both old and new information in the utterances comprising a dialog was discussed in Chapter VIII, Section D.3. The speaker expects the hearer to know the old information but to be unfamiliar with the new. Comprehension entails identifying the old concepts in memory and attaching the new information to them. Hence, identification of the old information in memory is an important part of the comprehension process. There are several syntactic devices for expressing old information (e.g., definite noun phrases, cleft

sentences, adverbials such as "too" and "still"). In this chapter we will be concerned with the processing needed to handle definite noun phrases since they are the most frequently used means of expressing old information. We will refer to the process of identifying the concept referred to by a definite noun phrase as "resolving the reference" or "resolving the definite noun phrase". Since context plays a crucial role in this identification process, a major concern of this section will be on using a representation of context to aid in resolving references.

The remainder of this section describes the role of context in reference resolution. Section B provides an overview of the use of a focus space partition of a network to represent context. Section C discusses several categories of definite noun phrase references and procedures for interpreting them. These procedures depend on the existence of a representation of focus of attention. The point of the section is to show the processing that must be done to build a representation of a particular definite noun phrase, given that noun phrase and a representation of the context in which it appears.

1. SENTENTIAL AND DIALOG CONTEXT

As in Chapter VIII, it will be useful here to divide definite noun phrase references into two categories: pronouns and nonproncninal definite noun phrases (DEFNPs). Although referring expressions in both categories depend on the context in which they occur for their

interpretation, the nature of this dependence is quite different in each case. Similarly, although some of the processing required for building interpretations of pronouns and DEFNPs may be shared, there is other processing that is unique to each of these forms of reference. Both the global dialog context and the immediate context of the preceding utterance play roles in interpreting each of these forms of reference, but the former is more important for DEFNPs, the latter for pronouns.

The major differences between these two kinds of reference stem from differences in the amount of information contained in the two kinds of referring expressions. DEFNPs contain more information in themselves than pronouns. The head noun of a DEFNP specifies the class of the object being referred to (in elliptical NPs and NPs with "one" as the head, the specification must be found contextually), and additional descriptive and distinguishing information is provided by modifiers. The global discourse context in which a DEFNP occurs plays a crucial role in resolving its reference. This context delineates the set of objects from which the object referred to must be distinguished. Sentential context, however, does not play a role in DEFNP resolution. In contrast, pronouns carry little information in themselves. They are really slot fillers and usually depend only on the sentential context in which they occur to provide most of the clues needed for identifying the referent. The exception to this -- pronouns that refer back over long pieces of discourse -- are discussed in the Section D.3 of Chapter VIII.

The relative role of sentential context in resolving DEFNPs and pronoun references can be seen by considering an example from Charniak (1972) and some variations of it. The original dialog is presented in Figure IX-1. The "it" in (7) can be resolved only when the context of "take ... back" is considered (and even then a large amount of inferencing must be performed; e.g., see Charniak; Hobbs, 1976).

- (1) Today was Jack's birthday.
- (2) Penny and Janet went to the store.
- (3) They were going to get presents.
- (4) Janet decided to get a top.
- (5) "Don't do that" said Penny.
- (6) "Jack has a top.
- (7) He will make you take it back."

Figure IX-1. THE KITE STORY

Note, however, that this "it" cannot be replaced by the DEFNP "the top". The problem stems from the fact that the context in which the utterance appears includes two tops, but use of the phrase "the top" implies there is only one. The sentential context of "take ... back" does not help eliminate one top as a possible referent when the DEFNP is used, as it does when the pronoun is. Finally, if instead of (7) the sentence were

"If you get Jack a top, he will make you take (it / the top)
back" ,

either "it" or "the top" may be used and the reference to the hypothetical top of the if-clause is clear. The difference between the use of "the top" here and in (7) is that here the if-clause sets up a new context in which there is only one top: the hypothetical one.

In many respects pronoun reference is closer to ellipsis, which will be discussed in the next chapter, than to DEFNP reference, and in a sense, the use of pronouns and ellipsis are duals. To see this, consider a sentence (S) composed of constituents A,B,C, i.e., assume that a context free part of a language definition rule for S is $S \rightarrow A B C$. Let a, b, c be respective instances of the particular phrase types A, B, C. Pronoun reference entails substituting a pronoun for one of these constituents; the remaining constituents are used to provide selectional restrictions on what the referent of the pronoun is. For example, in the 'sentence', "it b c", properties of b and c are used to find the object referred to by "it". Ellipsis, on the other hand, entails using only one of the constituents and, depending on context, to supply the others. So, if a' is also an instance of A, the 'sentence' "a'" in the context of the previous utterance, "a b c", may be expanded to "a' b c". Elliptical expressions can always be resolved in terms of the immediately preceding utterance. Elliptical DEFNPs (e.g., "the four by the door") and DEFNPs with the word "one" substituted for the head noun are like pronouns in that a slot (or a slot holder) is given, and the immediate sentential context and the preceding utterance are used to "fill out" the phrase, but they are like DEFNPs in the role played by the global dialog context.

2. THE INFERENCE PROBLEM

The simplest form of DEFNP resolution occurs when a phrase is matched with an object that has been described the same way previously in a discourse. This form occurs in the reference to a wrench in the sequence:

I bought a new wrench today.

The wrench is on the table.

However, restricting the use of DEFNPs to such cases results in rather boring discourse since it requires explicit statement of obvious facts. For example, the second sentence of the following sequence

Susan bought a car today.

The car has seats.

The seats . . .

is totally unnecessary and makes for awkward reading. Such redundant information usually is left out of a discourse. Comprehension then requires that the hearer be able to fill in the missing information from what he knows about the objects and actions being discussed. As a result, the resolution of DEFNPs often requires inferencing on the part of the listener.

There are two kinds of inferences that are needed for resolving DEFNPs. First, a reference may entail establishing additional properties of an object already in focus. Second, a reference may refer to an object that has been brought into focus only implicitly. As an example of the first case, consider the sequence

I took your coats to the cleaners.

The blue coat will be ready tomorrow.

To understand the DEFNP, the hearer must infer that one of the coats is blue. More frequently, an object once in focus may be referred to in more general terms than those in the description first used to bring it into focus. Resolving the reference entails establishing that the new description is true of the old object. In the sequence:

I bought a novel today.

The book ...

the fact that novels are books must be inferred.

The problem posed for resolution here is not the difficulty of the inferences themselves, but rather restricting the number of objects considered. That is, it is not the chain of inferencing that is the problem, but the number of times that chaining must be done. For example, in the preceding example, the inference chain relating novel and book is not long. The question is whether to look at all books and see which has been mentioned recently (doing the inferencing from book to novel), or to look at all objects mentioned recently and see which is a book (doing the inferencing from novel to book).

The second kind of inferencing required for DEFNP resolution arises because an object implicitly brings certain of its subparts into focus when it is brought into focus. For example, mention of "the living room" brings into focus items such as "the ceiling" and "the furniture". In the ensuing discourse, these objects may be referred to by DEFNPs. In the sequence:

E: Use the crescent wrench.

A: The handle is too long.

the phrase "the handle" can be resolved because the handle of a wrench is brought into focus when the wrench is. Parts of actions as well as objects may become focused in this way. For example, in the sequence:

E: Attach the pump to the platform.

A: Where are the bolts?

"the bolts" become focused because they are a part (namely the fasteners) of this attaching operation.

Chafe (1972, 1974) has pointed out the necessity for 'foregrounding' more than what is explicitly mentioned. His concept of 'being in consciousness' is similar to the notion presented here of being in focus in the discourse. The problem in handling this kind of inference is deciding how much information related to an object should get brought into focus when that object is. This issue is clearly related to the question of what goes into the 'frame' (Minsky, 1974; Winograd, 1975) for a concept. This kind of inferencing was not handled in the speech understanding system because of the lack of structure in the data base dialogs. In the task domain, the hierarchical structure of the task and the correspondence between task structure and dialog can be used to provide both explicit and implicit focusing (see Grosz, 1977).

B. THE FOCUS SPACE ENCODING OF CONTEXT

Several of the preceding examples, as well as examples presented in Chapter VIII, point out the need for a representation of the global discourse context in which an utterance appears. This section provides a brief overview of an approach developed in the framework of a semantic network knowledge representation. The representation chosen has several distinguishing features. It highlights that part of the semantic network relevant at a given point in a dialog, grouping together those concepts which are in the focus of attention of the dialog participants. The ability to link the context representation with representations of surrounding task situations also is provided. The representation itself is structured so that the structure of the dialog (see Chapter VIII, Section D.2). can be mirrored and used in discourse processing. Finally, the representation has the potential for extensions in two directions closely related to context: focusing on different attributes of the same object under different circumstances and forgetting information no longer relevant to a discourse.

1. EXTENDING THE NOTION OF PARTITIONING

To encode context, or focus of attention, we will extend the notion of partitioning of networks described in Chapter V, Section D, and in Hendrix (1975a,b). We will now allow a network to be partitioned along multiple dimensions. Each partition will be independent in the sense that the spaces on which a node or arc lies in one partition

neither determines nor depends on the spaces it lies on in any other partition. In particular, in addition to partitioning the network along 'logical' lines, we will partition it along 'focus' lines. The logical partition will remain as described previously; every node or arc lies on at least one space in the logical partition. In addition, nodes and arcs may lie on spaces in the focus partition.

Network partitioning will be used both for its ability to separate entities into spaces and for its ability to relate different spaces hierarchically. We note here that although we are using a network representation, the use of partitioning of memory structures for the purpose of reflecting focus of attention is a general one and may be used in other representation schemes.

The focus partition is not a partition in the mathematical sense, since nodes and arcs may lie on any number of focus spaces, or on no focus space at all. At any point in a dialog, one focus space will be 'active', but several may be considered 'open'. The active focus space will reflect the focus of attention of the dialog at that point in the dialog. The open focus spaces will reflect previous active spaces that contain some unfinished topics and hence may become active again; they are possible areas for the dialog to shift back to. Hence, the focus partition enables the portions of the network that are relevant to a dialog to be spotlighted and a trace of the spotlighting to be kept.

When the same object enters the dialog twice, in two different subdialogs (e.g., a tool used in two distinct subtasks), the node corresponding to that object will appear in two distinct focus spaces. If different aspects of the object are focused on in the two subdialogs, different relations in which the object participates will be in the two focus spaces. Hence, focusing also allows the particular way of looking at a concept that is germane to a given point in a dialog to be spotlighted.

The main reason for providing the ability to focus on different attributes of an object is to allow differential access to the properties of the object, and hence to the facts that may be derived about that object. Using the arcs in focus for differential access does not rule out considering a concept differently than it has already been portrayed. Instead, it orders the way in which aspects of the concept are to be examined in looking for new (to the dialog) information about the concept.

Differential access is important for actions as well as physical objects. For example, when quilting is considered as a kind of sewing, the subactions of cutting and pinning are directly accesible, but when quilting is considered as a social gathering then the subactions of talking and eating are more important, and more accessible.

2. MATCHING IN FOCUS

The basic process involved in resolving DEFNPs is matching structures in the memory representation. In the case of a system with a semantic network knowledge base, the problem is that of finding a network structure matching the structure of the noun phrase. This section describes the role of focus spaces in this matching process. The presentation assumes that items get moved in and out of focus as needed. The shifting of focus in the data base dialogs is basically linear with time. As a result, the implementation in the speech understanding system assumed a linear shift in focus. Concepts mentioned in one utterance were moved into focus and kept there until a small fixed number of other utterances had been processed. This is a minimal use of focus. It corresponds directly to the linear history and working space models in other systems (see Norman et al., 1975). As pointed out in Chapter VIII, Section D.2, the task dialogs are much more structured than the data base dialogs. As a result, much more sophisticated handling of focus can be done for the task dialogs. In particular, focus spaces can be used to relate shifts in focus to the dynamics of the task (see Grosz, 1977).

In the speech understanding system, this matching procedure is performed by the deduction component (described in detail in Chapter XII). The deduction component is called with a QVISTA (question vista) and a KVISTA (knowledge vista). The QVISTA is a set of spaces containing a piece of network for which a match is sought. The KVISTA

represents the set of all knowledge in which the match is sought. In the process of arriving at a match, the deduction component binds the arcs and nodes in the QVISTA to arcs and nodes in the KVISTA. When a match is found, the node in the KVISTA bound to the node corresponding to the head noun of the DEFNP in the QVISTA represents the object referred to by the DEFNP.

The focus representation can aid the deduction component in two ways: by focusing on particular nodes, and by focusing on particular arcs from those nodes. By focusing on certain nodes, the deduction component can be constrained to consider only objects germane to the dialog. Focusing on an arc guides the deduction component in establishing properties about nodes being matched. We have some experience in directing the deduction component to match nodes in focus; this will be described in some detail. Details of focusing on arcs is described in Grosz (1977). For the resolution of DEFNPs (the discourse use of the deduction component), the QVISTA is the vista constructed at parse time for the DEFNP and the KVISTA is the knowledge space. In addition, matching in focus currently entails passing the deduction component a focus vista and a list of nodes to be matched in focus. Matching in focus means the bindings made by the matcher are restricted to the items in focus; however, deductions can be made using any information in KVISTA. The focus vista represents the set of nodes and arcs considered to be 'in focus'. In DEFNP cases the node corresponding to the head noun of the NP is forced to be in focus. This corresponds to saying the referent of a DEFNP is sought (first) in focus.

The simplest use of a focus match is to arrive at the correct match. Consider the situation portrayed in Figure IX-2. There are several wrenches: W1 is a box-end wrench that is in focus FS1; W2 is a box-end wrench in focus FS2; W3 is an open-end wrench also in focus FS2; W4 is another open-end wrench not in focus at all. There is another object, O1, with a box-end. Space q.w1 of Figure IX-3 shows the QVISTA corresponding to the DEFNP "the wrench". If the deduction component were asked to find a match without focus for this query, any of the W nodes would do. This corresponds to the fact that, without focus, the phrase "the wrench" is four ways ambiguous. However, if the deduction component is provided with QVISTA q.w1 and focus vista FS1 (and the node QW1), it will find that W1 is the only match. In arriving at this solution, it may consider H1 but will discard this possibility when realizing that hammers and wrenches are mutually distinct subsets of tools. The attempt to match "the wrench" in focus space FS2 will result in both W2 and W3 matching, reflecting the fact that, for the discussion at that point, two wrenches were relevant, and the phrase is ambiguous.

The second use of focus is to reduce the amount of computation done in arriving at a match. Most QVISTAs are not as simple as q.w1. Space q.w2 of Figure IX-3 shows the QVISTA of the DEFNP "the box-end wrench". Consider what could happen in an unconstrained match. The deduction component would consider all of the nodes with 'e' arcs to wrenches or all of the nodes with 'endtype' arcs to BOX-END (depending

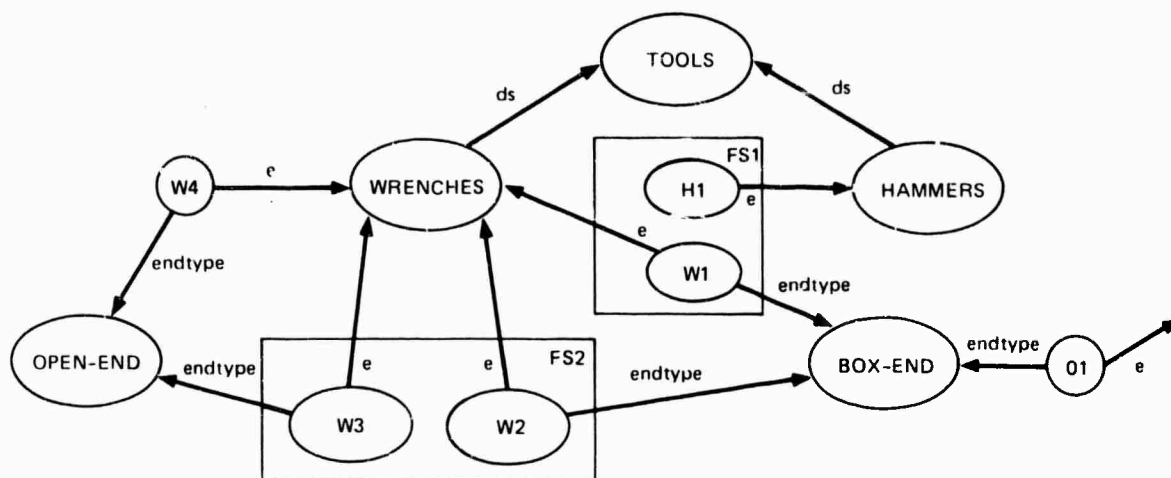


FIGURE IX-2 A SIMPLE KVISTA WITH TWO FOCUS SPACES

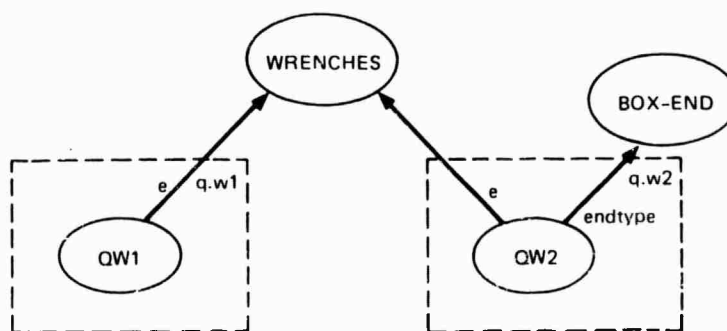


FIGURE IX-3 QVISTAS FOR "THE WRENCH" AND "THE BOX-END WRENCH"

on which set is smaller) until it tried W1 or W2. If asked for a second match, it would (eventually) determine that both W1 and W2 matched. Note that in the worst case this could entail one node and two arc bindings for each of the nodes in the set selected.

The constrained match is able to avoid all this hunting. If focus space FS1 is used, only nodes H1 and W1 are considered (as before). With focus space FS2 as the constraint, both W3 and W2 would be considered but W3 eliminated. In the worst case, one set (one 'e' arc and one node) of unnecessary bindings would be made.

The advantage of focus spaces in terms of the number of bindings attempted depends on having fewer items 'in focus' at any one time than are in a typical set in the knowledge vista. But this is exactly the purpose of focus -- to highlight those few items relevant to a given point in the dialog.

A similar computational advantage is gained when deduction is necessary to achieve a match; i.e., when theorems -- information stored in the net as general rules applicable to whole sets of concepts -- must be applied. To illustrate such a match, consider the KVISTA of Figure IX-4. Here the set of wrenches has two subsets, 'B-E', the set of all box-end wrenches, and 'O-E', the set of all open-end wrenches. Finding matches for the QVISTA q.w1 in this KVISTA entails following the e and s chain from, say W1, to 'WRENCHES'. This process is the simplest form of deduction. Even assuming that all of the elements of 'WRENCHES' were represented as elements of sets that were subsets of 'WRENCHES', finding some match for QW1 would not be too complicated. (We still have the problem of finding the right match, given that there is more than one).

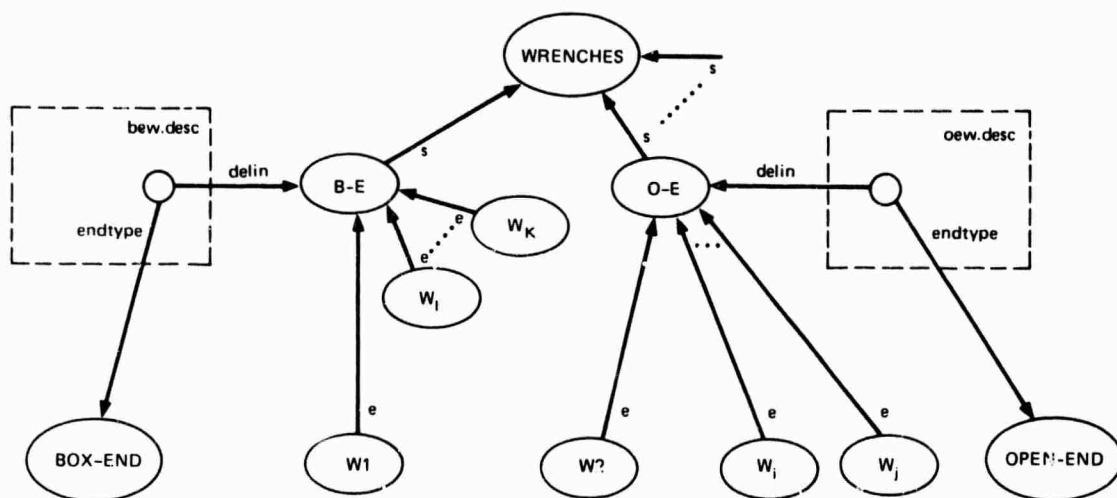


FIGURE IX-4 A KVISTA WITH THE SET OF WRENCHES DIVIDED INTO SEVERAL SUBSETS

The situation gets more complicated when we consider finding a match for QVISTA q.w2. For purposes of this discussion, assume that 'WRENCHES' has fewer elements than 'BOX-END'. (This assumption simplifies the discussion, and is reasonable, considering that objects other than wrenches may be classified as "box-end".) The match for 'QW2' proceeds by considering all nodes with 'e' arcs to 'WRENCHES'. The 'e' arcs are all implicit in this case; they must be derived by following e-and-s chains. For each element of wrenches proposed as a match for 'QW2', the deduction component attempts to establish an 'endtype' arc to 'BOX-END'. In particular the delineating element descriptions for 'B-E' and 'O-E', contained in the logical spaces

bew.desc and oew.desc, respectively, represent applicable theorems. If 'W2' is tried as a match, the deduction component will establish the e-link to wrenches. The relevant theorem in this case is in the box labeled oew.desc. A tentative endtype arc from W2 to 'OPEN-END' will be constructed and only then will the deduction component realize 'OPEN-END' is not 'BOX-END', and hence W2 will not match. In general, there will be many nodes like W2 that appear to be candidates but do not match, and many theorems that will apply. The work done before considering W1 may be extensive. By constraining the search to nodes in focus, a considerable reduction can be achieved. In particular, suppose there are n elements of wrenches, of which m are box-end. In the worst case, $n-m$ incorrect bindings and unnecessary theorem invocations will be computed. (Even the average of $n-m/2$ is significant if n is much larger than m .) However, if there are k nodes in focus of which W1 and W2 are the only ones that are wrenches, then at worst k wrong bindings and one unnecessary theorem will be tried. Since the cost of a theorem is typically much greater than the cost of establishing a binding -- even following e and s chains -- and typically $k \ll n$, the savings are substantial.

A further computational advantage of focus may be seen from the modified version of the wrenches situation portrayed in Figure IX-5. A focus space, FS, has been added. In addition to containing the nodes W1 and W2 (which we are assuming are already in focus), it contains the 'endtype' arc from W1 to 'BOX-END' and an explicit 'e' arc

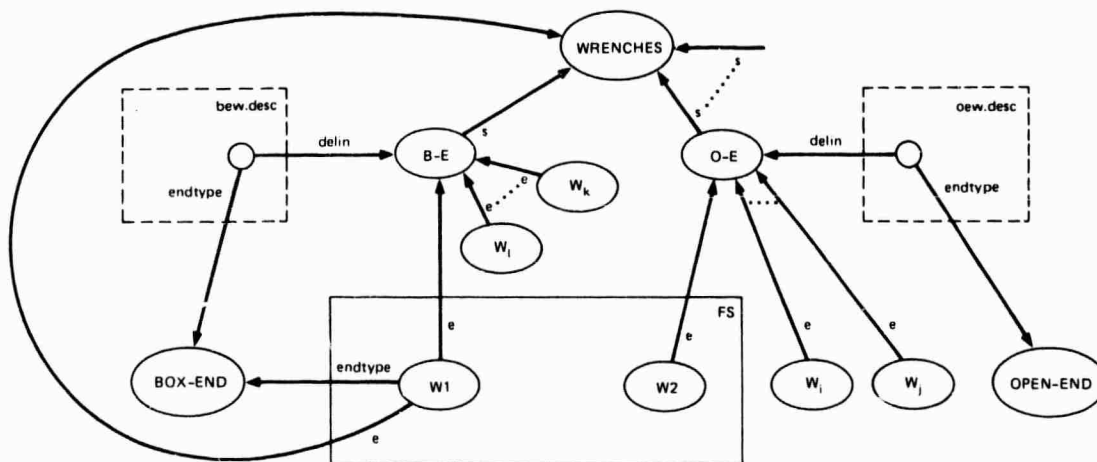


FIGURE IX-5 THE WRENCHES KVISTA WITH FOCUS ADDED

from 'W1' to 'WRENCHES'. Any matches sought for "the box-end wrench" while the focus is FS will be able to take advantage of this explicitly stored information. This information is redundant; ideally, the arcs would disappear as soon as FS was closed. Incorporating this feature would have the desirable results of both having the information available when it was relevant and allowing it to be 'garbage collected' or 'forgotten' after it ceased to be relevant. This is one possible extension of the focus space mechanism.

C.. DEFNP RESOLUTION IN CONTEXT

The typical noun phrase has several constituents. For the purposes of this discussion we will consider the structure of an NP to follow one of the patterns

- (1) (DET/QUANT)[NUM] NOM
- (2) (DET/QUANT)[NUM]
- (3) NUM

(These rules do not correspond to the actual rules in the SRI speech understanding system language definition. However, the grouping is convenient for purposes of discussing discourse processing.) In this notation, the slanted line indicates a choice of one or the other constituent, parentheses are used for grouping, and brackets indicate an optional constituent. DET is the category containing determiners; it contains words such as "the", "this", and "which". QUANT is the category of all quantifiers; e.g., "all", "any", "some". NUM is the set of number expressions; e.g., "one" and "three hundred fifty." NOM, the set of nominal expressions, contains unmodified nouns, premodified, postmodified nouns, and nouns both pre- and postmodified. Respective examples of such NOMs are "wrench", "box-end wrench", "wrench with the red handle", and "box-end wrench with the red handle." Form (1) is the form of NP with which we will be most concerned; forms (2) and (3) are both elliptical and are not handled in the speech understanding system.

There are many syntactic and semantic problems associated with parsing and building representations for the group of phrases in the category NOM. Some of these are discussed in Chapter VIII, Section B.

For example, it takes semantic knowledge to determine the difference between "the big ship" and "the German ship". For the purposes of this section, these problems can be ignored. We will assume that any NOM has been checked syntactically and that a semantic representation has been built for it. It is only when looking for the concept described by the NOM that discourse processing is really needed.

1. FROM SEMANTICS TO DISCOURSE

The semantic interpretation for the NOM constituent of a noun phrase encodes the relationships among the concepts that are conveyed by the constituents of the NOM in the underlying knowledge representation. In essence, it provides a representation of the typical item described by the NP. For example, the representation for "American sub" in the partitioned semantic network notation is shown in SPACE P1 of Figure IX-6. Note that the 'ownership' relation conveyed by "American" in this particular construction is represented in this network structure. The discourse component contributes to building an interpretation of an NP only if the determiner or quantifier for the NP indicates definiteness. [The elliptical form (3) of the NP constituent structure implicitly conveys definiteness.] The basic problem for the discourse routines is to locate the object or set currently in focus which corresponds to the description in the NOM part of the NP. When an instance of NUM is included in the NP, discourse processing is influenced only insofar as a check on the set found is required to be

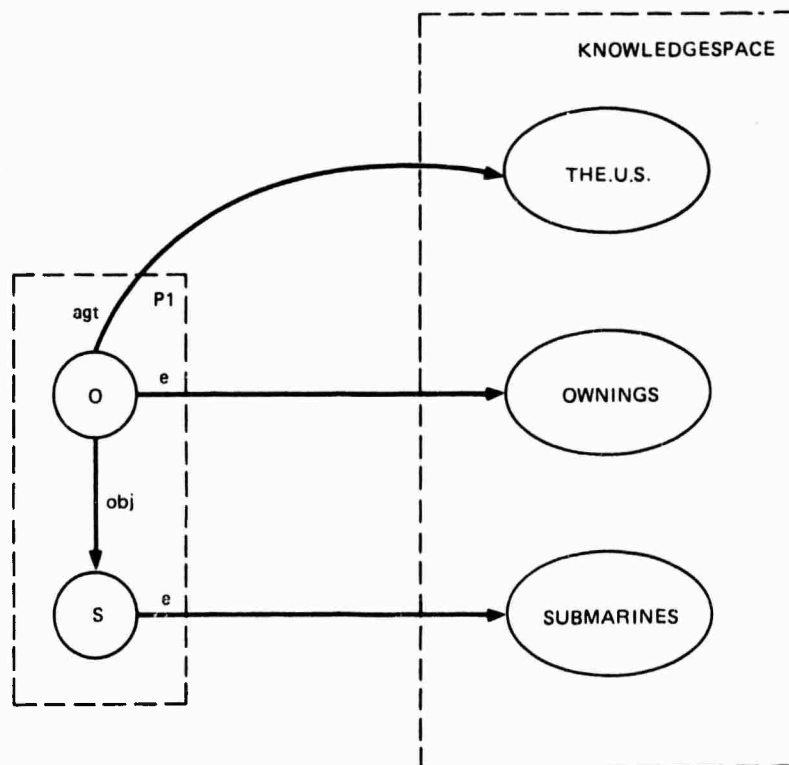


FIGURE IX-6 PARSE LEVEL SEMANTIC NET REPRESENTATION FOR "AMERICAN SUB"

sure the set has the correct cardinality. "One" is an exception and is treated like "a" rather than other, plural, NUMs. For the NP, "the American sub", an individual submarine owned by the U.S. must be found in focus. For the NP, "all six American subs", a set of (exactly) six subs, all owned by the U.S., must be found.

2. INTERPRETING COMPLETE NPS

The deduction component, when augmented for focus matches as described in the preceding section, performs the central function in the process of interpreting complete NPs. Given the semantic interpretation of a DEFNP and the current focus vista, it determines which, if any, object in focus matches the DEFNP. Note that the first kind of inferencing discussed in the overview occurs at this stage of the processing. The deduction component, in determining whether a given object in focus is the referent of the DEFNP, follows the subset hierarchy and deduces information from theorems in the network. The restriction of the search to the focus space is crucial; generally, the number of objects in focus is quite small and contradictions (e.g., if the candidate focus space node and the node corresponding to the head of the DEFNP are elements of mutually exclusive sets) can be reached quickly for many of the objects. At present, this matching procedure is carried on depth-first. In the limited data base domain for which resolution has been done, this strategy is sufficient. A parallel search has the advantage of finding the match more quickly, on the average. However, it is still necessary to establish that no other object matches in order to rule out ambiguities.

a. SINGULAR NPS

If the DEFNP is singular and a match is found, the node matching the node corresponding to the head noun is the referent. The

only further processing is to check that all relevant relationships are in focus. This process is described below in Section C.3. If a match is not found, one of three possibilities still exists (assuming the NP can be resolved!): the object may be unique (e.g., "the sun"); the DEFNP may refer to an object implicitly, but not explicitly, in focus (the foregrounding problem); or the DEFNP may contain a genitive or a modifier containing new information (e.g., the DEFNP, "John's car" when John is in focus but his car is not).

The uniqueness check requires determining whether more than one object fitting the DEFNP description exists. This check is done after the focus space check, because context may in fact overrule the usual uniqueness conditions. The phrase "the sun" in the sequence,

Mary has a beautiful sunset picture.

The sun is teetering above the mountain.

refers to the image of the sun in the picture, not the real sun; i.e., the sunset picture creates a context with a special sun. For relational DEFNPs, a unique result always is obtained, and the focus mechanism is not used.

References to objects implicitly in focus will not be considered here. Extensions to current routines necessary for handling modifiers and genitives are discussed after the section on plural NPs.

b. PLURAL NPS

A plural DEFNP may create a new set by grouping together objects already in focus. In the sequence,

You will need the wrench, the screwdriver, and the hammer.

Should I put those tools in the toolbox?

the DEFNP "those tools" (note that the pronoun "them" could also have been used) refers to the set of three individual tools in focus. The set itself, however, does not exist as a node in the network.

The resolution routines handle this problem by looking for individual objects in focus that satisfy the DEFNP. (Since the deduction component does not yet handle sets, this is the only kind of plural resolution that is done in the current system implementation.) If it finds more than one such object, a new set is created and added to focus. Future references to the set, either by pronoun or DEFNP, will lead the deduction component to find the set.

c. MODIFIED NPS

Modifiers may be used in three ways. The simplest case is the use of modifiers to select among individual objects in focus. This case entails a straightforward match (although some inferencing may be required). Problems arise only when a modifier is used to select an element of a set in focus (when the individual elements of the set are not in focus) or when modifiers are used to supply new information about some objects in focus. An example of the former occurs in the sequence,

A high school class came to visit the hospital.

The brightest student . . .

The DEFNP "the brightest student" singles out an element of the high school class. An example of new information being added by the DEFNP occurs in the third sentence of the sequence,

Jane got some books today.

They're on the coffee table.

The new book by Haley is on top.

The DEFNP "the new book by Haley" singles out one of the set of books. The information that Haley wrote it and that it is new is introduced by the DEFNP.

The existing DEFNP routines will fail to find a match in these two cases. What needs to be added is the ability to remove modifiers from the DEFNP until a (unique) match can be found, i.e., to try successively less restrictive matches. The use of network partitioning to reflect the parse structure in the semantic interpretation of phrases provides a means of removing modifiers from DEFNPs. It also is crucial for handling ellipsis (see Chapter X).

d. GENITIVES

Genitives may cause two kinds of problems. First, the genitive may be used to supply new information about an element or set already in focus. When used this way, a genitive is like any other modifier. As an example, consider the use of the DEFNP "Peter's car"

when a set of cars, one of which is owned by Peter, is in focus. Assuming "Peter" is unique, ownership by Peter can be asserted of one of the cars. This use of the genitive may be handled exactly like other modifiers.

The second and more interesting problem arises when a genitive is used to supply the old information in a phrase. That is, the genitive constituent of a DEFNP may refer to an object in focus, while the object referred to by the complete DEFNP may not be in focus. For example, assume a focus in which there are two people, a boy and a girl. Then the phrase "the boy's mother" is unambiguous and resolvable because the boy is in focus and mother is a unique relation. That is, even though there is no mother in focus, there is a boy in focus, and the relation conveyed by the genitive can be used to determine, via the link to the boy, which person is being referred to. Note that foregrounding is not the issue here; the phrase "the boy's school" is equally resolvable. In a sense, a DEFNP with a genitive has two heads: the head of the genitive, as well as what is usually considered to be the head noun. For this reason, if a DEFNP with a genitive cannot be resolved, the processing must proceed in two stages. If the whole DEFNP cannot be resolved, the genitive alone must be considered. The genitive must be resolvable. If the remainder of the NP is not resolvable in focus, then the genitive relationship must be used to determine uniqueness. For example, if some boy is in focus, "the boy's school" is resolvable by accessing what is known about the boy and determining if school is unique.

e. QUANTIFIED DEFNPS

The processing of quantified DEFNPs is the same as that for unquantified plural DEFNPs except for the consideration of a generic interpretation. There are several cases, depending both on the particular quantifier used and on whether the optional NUM (number) constituent is present in the DEFNP. If the optional NUM is included in the NP, then the generic is never intended. Instead, it is always the case that a local set must be found, with the correct cardinality, over which the quantification holds. To see this, contrast "All subs have beams over 30 feet." with "All five subs have beams over 30 feet." In the first utterance, the generic interpretation is clearly preferred. In the second, the generic interpretation is not possible; a local (nongeneric) set must be identified. Although this construction (i.e., QUANT NUM NOM) can be used with "any" (and to a lesser extent with "some") as the quantifier, its most common use is with "all"; hence, "all" is the only quantifier allowed in this construction by the current language definition. However, the nongeneric meaning of the construction holds for the other quantifiers as well.

For constructions not including NUM, the question of interpreting a phrase generically depends on the quantifier. Quantifiers implicitly conveying a set of size two "both", "either", "neither" are never generic. There must be a local set matching the NOM and with exactly two members, for these quantifiers to be meaningful. In contrast, "all" always conveys the generic; the construction "all

the x" is used to limit the restriction of "all" to some local set.*

"Some" and "every" also tend to convey the generic, but less strongly than "all". The heuristic we currently use is to assume the generic use if the NOM is unmodified and otherwise to check first for a local set meeting the specifications of the NOM. If no such set exists, then the generic is assumed. There are clear counter-examples to this rule; e.g., in the utterance, "Some tall trees are killed by lightning", the generic is intended even if there is some particular set of trees in focus. This case is not currently handled by the discourse routines. For the remainder of the quantifiers a local set is checked for first. If none exists, the generic is used.

3. AUGMENTING FOCUS

Once an utterance has been parsed, the concepts occurring in the utterance must be added to focus. When the matcher returns a match for a DEFNP, more information than the node corresponding to the NP is returned. Recall that the semantic interpretation for an NP is a set of nodes and arcs encoding the relationships expressed in the NP. For the DEFNP, "the red box-end wrench", the encoding, shown in Figure IX-7, includes an element arc to the set of wrenches, an endtype arc to box-end, and a node representing the relationship of being colored red

At first there seems to be some ambiguity between expressions involving "all" meaning "all in the computer knowledge base" and "all in the world". However, this ambiguity can be seen only from a frame of reference outside the computer model. Inside, the two are, by definition, equivalent.

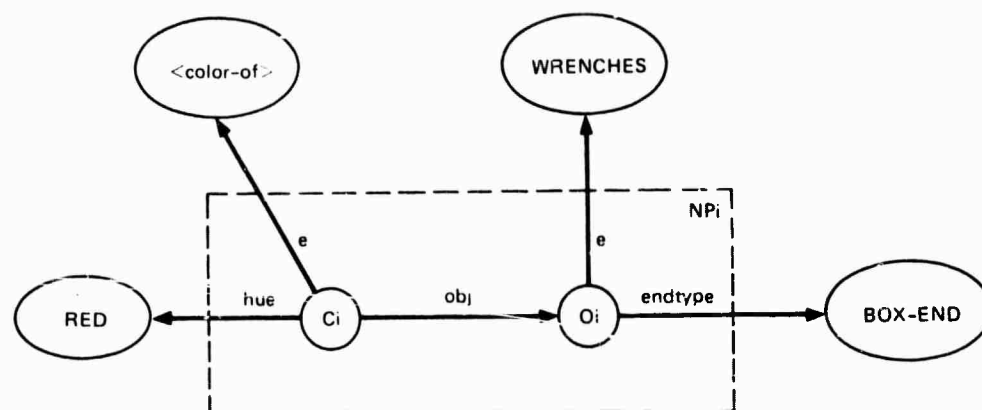


FIGURE IX-7 SEMANTIC REPRESENTATION FOR "RED BOX-END WRENCH"

("red" and "box-end" are represented differently because the color of a wrench can change but the endtype cannot). To see what happens when a match is made for the DEFNP, consider the focus setup of Figure IX-8. Focus space FS_i contains three wrenches. Two of them are in set BEW, a subset of all wrenches that includes only box-end wrenches. Of these, one is red, the other green. In the process of identifying "the red box-end wrench" with W_1 , two new arcs will be created: an element arc from W_1 to wrenches and an endtype arc from W_1 to BOX-END. These are the result of various network deductions using information in the net of Figure IX-8. In addition to returning the correspondence between Node O_i and W_1 , the deduction component returns correspondences between C_i and CW_1 and a set of arc correspondences. The latter includes the correspondence between the e-arc out of O_i and the newly created e-arc from W_1 and the correspondence between the endtype arc from O_i and the newly created endtype arc.

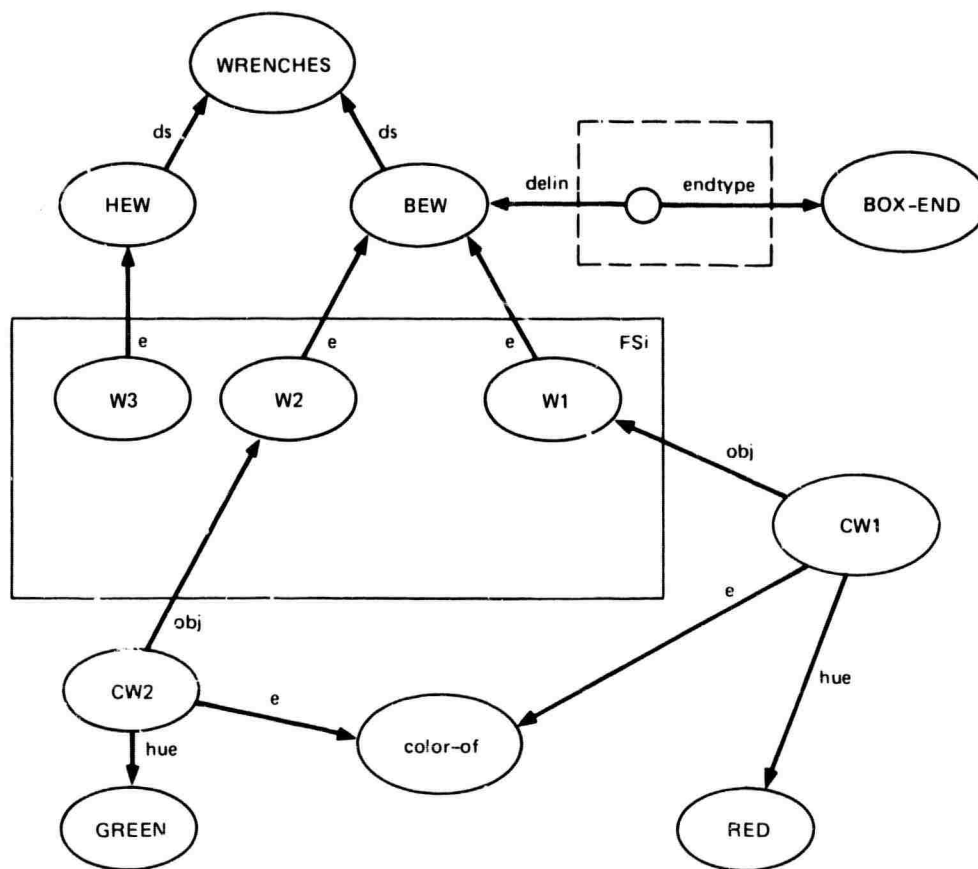


FIGURE IX-8 ORIGINAL FOCUS SPACE

Updating the focus space entails moving each node or arc that corresponds to a node or arc in the NP space to the focus space. The result, for this NP, is shown in Figure IX-9.

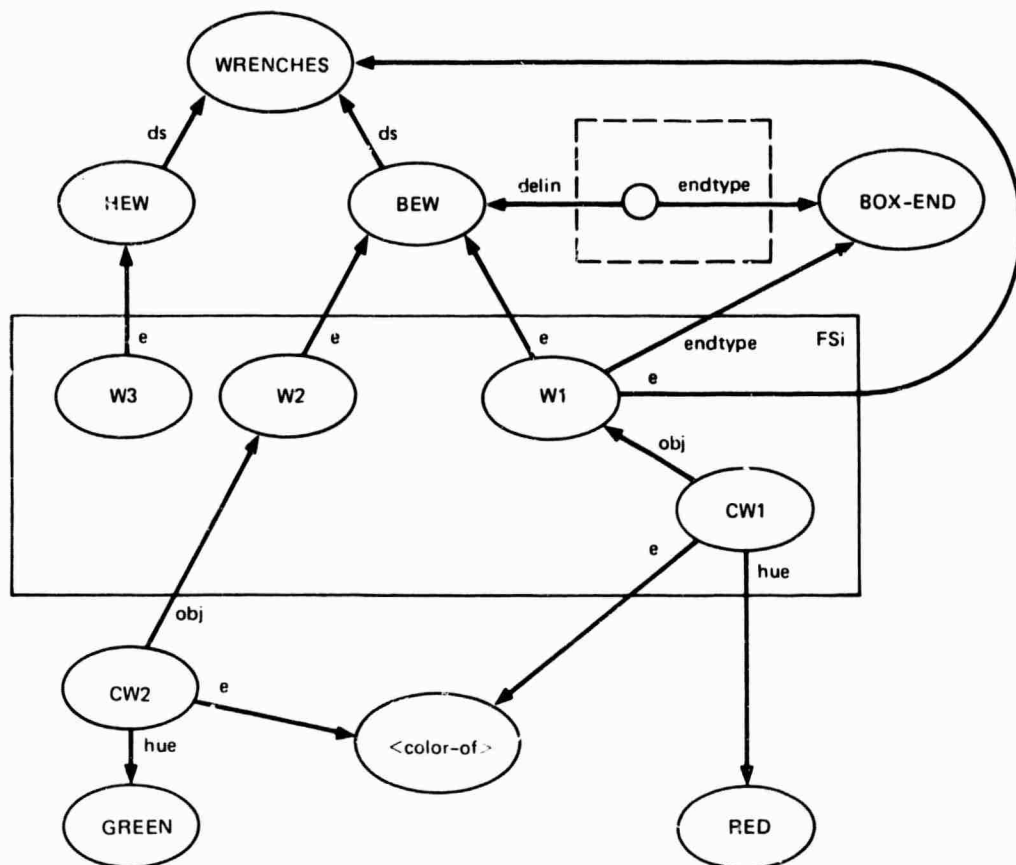


FIGURE IX-9 NEW FOCUS SPACE

X ELLIPSIS

Prepared by Barbara J. Grosz

CONTENTS:

- A. Overview
- B. Slot Determination
 - 1. Syntax
 - 2. Semantics
- C. Completing the Utterance
- D. Elliptical Relational Noun Phrases
- E. Limitations and Extensions

A. OVERVIEW

The context of an utterance not only provides the semantic framework for resolution of definite noun phrases, but also the syntactic and semantic framework for interpreting elliptical utterances. Ellipsis refers to the use of incomplete grammatical units in a discourse (the items left out are 'elided'). Although such a unit is ill-formed by itself (in the traditional competence grammar sense), if the context in which it appears supplies the elided items, it is well-formed. For example, the utterance,

The crescent wrench

is an incomplete sentence, but if it appears in the context of the question,

What tool are you using to loosen the bolts?

then it is easy to construct the complete sentence it is meant to convey, namely,

I am using the crescent wrench.

"The crescent wrench" is an example of ellipsis at the sentence (or clause) level. Ellipses may occur at the noun phrase or verb phrase level as well. The following sequence is an example of noun phrase ellipsis:

Which box should I use for the tools?
Only the largest will hold all the tools.

Verb phrase ellipsis is shown in the following sequence:

Has the pump been tightened down?
No, but the motor has been.

Halliday and Hasan (1976) present many examples of each of the three forms of ellipsis (clausal, noun phrase, and verb phrase) and the means by which they can be used to link successive sentences in a discourse. The emphasis of this section will be on ellipsis at the sentence level, because it is the form of ellipsis that occurred most frequently in the dialogs.

It is important to note that if the constituents missing from an elliptical phrase can be found at all, they can be found in the immediately preceding utterance. If there is a sequence of three utterances u_1 , u_2 , and u_3 , then the structure of u_2 can be matched against u_1 , and u_3 can only be matched against that of u_2 , but the presence of u_2 precludes matching u_3 against u_1 . In the long sequences

of questions to be discussed shortly, although it appears that u3 is patterned on u1, in fact u2 is expanded to a form similar to u1 and then u3 is patterned on u2. It is in this regard that ellipsis is a more local phenomenon than reference. Only the immediate focus of an utterance contributes to expansion of any elliptical phrases in the utterance. The global discourse context is not significant.

The process of building an interpretation of an elliptical phrase entails two steps once the ellipsis has been detected. First, the items missing from the utterance must be found in the preceding utterance (or, equivalently, the slot the elliptical phrase fills in the preceding utterance must be determined). Second, a complete phrase must be built using the elliptical phrase and the missing constituents found in the previous (old) utterance. In the remainder of the discussion, the first step will be referred to as "determining the slot", the second as "expanding the utterance".

The use of ellipsis was different in the task dialogs and the data base dialogs (see Chapter VIII, Section D.4). In the task dialogs, elliptical utterances appeared as responses to questions. In the data base dialogs, elliptical utterances were used in long sequences of questions. For purposes of building an interpretation of an utterance, the difference has most impact on the slot-determining phase of processing. In the question-and-answer pairs of the task dialogs, the slot filled by the elliptical answer is often marked in the question by a WH-phrase. Determining the slot filled by the ellipses in the

question sequences of the data base dialogs is not so straightforward; syntactic and semantic clues must be used as explained below. Expansion of the utterance entails similar procedures in the two domains, but some preliminary transformations are required for the ellipses in the task domain (see Robinson, 1975a).

The remainder of this section concentrates on capabilities in the discourse component for handling the elliptical utterances that occurred in the data base dialogs. The procedure for interpreting the elliptical utterances (EU) in the context of the preceding pattern utterance (PU) will be presented. In question-and-answer sequences, both the answer following a question and the next question itself may be elliptical. The PU for an elliptical answer is the preceding question. Expansion of this elliptical answer requires many of the same transformations as the elliptical utterances in the task dialogs. The PU for an elliptical question also is the preceding question, which is really two utterances back. This treatment is actually equivalent to using the immediately preceding utterance, the answer, since its structure corresponds directly to that of the question. The two utterances differ only in that one is marked as a question.

We limited the range of elliptical expressions we would handle in the speech-understanding system to noun phrases functioning as complete sentences, as in our initial example. This set covered most of the ellipses encountered in our protocols. More importantly, allowing more extensive noun phrase and verb phrase ellipses would have meant greatly

increasing the alternatives considered for these lower level constituents during the interpretation of an utterance. Expanding an elliptical phrase is a relatively expensive operation when compared, for example, with syntactic checks or semantic case checks. Doing it at the utterance level seems worth the cost since complete utterances are relatively infrequent compared with other constituents being proposed and found. If we had been working with error-free test input rather than speech, the overhead requirements would have been less extreme and other forms of ellipsis might have been allowed. Extensions and modifications needed to do more complete ellipsis handling are described in Section E.

B. SLOT DETERMINATION

1. SYNTAX

Syntax plays a major role in determining the slot filled by an elliptical utterance (EU). Usually, for an EU to make sense there must be a structural unit of the same type in the pattern utterance (PU). (This is not completely true: there may be an unfilled slot in the syntactic pattern for the PU that the EU fills. This case, and extensions to the algorithms for handling it, are discussed in Section E.) In addition to defining the category of phrase an EU can match, syntax also provides filters on the basis of definiteness and syntactic role.

If an EU consists solely of a noun phrase (NP), the determiner of that NP must match the determiner of the slot phrase in the PU. If the NP of the EU is definitely determined, it can match only definite NPs in the pattern; if it is indefinite, it can match only indefinitely determined phrases. The sequence PU - EU1 is fine, but PU - EU2 is awkward.

PU: Does England own a submarine.
EU1: A destroyer?
EU2: The destroyer?

It is possible to have a sequence of questions with indefinite NPs culminating in a definite NP, but this is an exceptional case; it occurs only when the definite NP refers to some truly unique object, or the questioner and answerer are playing a game. The following sequence showing an interchange between two people is an example of the former:

P1: Do you know what John got at the auction?
P2: Was it a document?
P1: Yes.
P2: An old one?
P1: Yes.
...
P2: The Constitution? / A copy of the Constitution?

The question-answering dialogs of the game "20 Questions" are an example of the latter. The same phenomenon happens with plurals. So the sequence PU - EU1 is fine, but PU - EU2 is not.

PU: Does England own any submarines?
EU1: Any patrol boats?
EU2: The patrol boats?

One can construct situations in which EU2 is reasonable, but again the set denoted by the NP must be unique. So, for a data base in which

there was only one set of patrol boats (and these are a subset of submarines), the sequence PU - EU2 might be acceptable. This use of the definite at the end of a series of indefinites is sufficiently rare that we have not modified the algorithm to handle it.

The parallelism of definites and indefinites is most clear when we consider utterances with two NPs that differ only in definiteness. Contrast the two sets of question sequences.

PU: Did the cat hurt a bird?
EU1: The dog?
EU2: A mouse?

PU: Did the cat hurt the bird?
EU1: The dog?
EU2: A mouse?

Without any preceding context, in the first sequence both EU1 and EU2 are unambiguous; the NPs match the correspondingly determined NPs. In the second sequence, EU1 is ambiguous; it could either be a question about the cat and the dog or one about the dog and the bird. The preference is to resolve the ambiguity on a semantic basis, but there is clearly some confusion that does not arise in the first sequence. Utterance EU2, in the second sequence, really does not make sense without some imputed context. Even then, there could be an ambiguity similar to the one for EU1.

The algorithm for determining the slot filled by an elliptical utterance uses the parallelism of determiners to filter out phrases to be considered as matches. The determiner of each NP in the PU is checked. If it is the same as that of the NP constituting the EU, then

the NP is a candidate for a match; the slot it fills is a candidate slot for the EU.

A problem arises when considering EUs consisting solely of nominals -- NPs without any determiners. Some default determiner must be chosen for the EU so that the filtering process can be done. The default currently used is definite for singular NPs and indefinite for plural NPs. This treatment is adequate for the kinds of questions in the data base domain seen in the following three examples from the data base protocols:

PU: What is the length of the Ethan Allen?

EU: Draft?

PU: Does Britain own any submarines?

EU: Patrol boats?

PU: Does the U.S. own the Ethan Allen?

EU: George Washington?

In general, however, there are cases that do not work undetermined:

PU: Did you drive the Cadillac today?

EU: Volkswagen?

"Volkswagen" alone is just not enough; "the Volkswagen" is. This pair actually points up the idiosyncratic nature of the preceding pair. Other nouns require no determiner and can be matched by other undetermined nouns or by definitely determined ones:

PU: Did he write about pollution?

EU: Ecology?

EU: The environment.

The syntactic role of a noun phrase is important in choosing between candidate slots that are filled by phrases which are otherwise semantically and syntactically equivalent. Consider the sequence:

PU: Is the Ethan Allen longer than the George Washington?

EU: The Churchill?

The EU is ambiguous since "The Churchill" could replace either "the Ethan Allen" or "the George Washington". However, both "Is the Churchill" and "Than the Churchill" are unambiguous. In each case a syntactic role is assigned to "the Churchill" that can be used to eliminate one of the two candidate slots.

In summary, syntax is used to limit the candidates considered for finding slots of NPs serving as EUs. First, only NPs with matching determiners are considered. If there is more than one candidate, syntactic role is used to eliminate choices. If at either step of the process there are no candidates, there is the option of relaxing syntactic constraints. This option was not pursued in the speech understanding system because of the need to restrict, rather than increase, potential interpretations.

2. SEMANTICS

Although syntactic restrictions often eliminate all but one choice, there are cases when an appeal must be made to semantic attributes of the phrases filling candidate slots in the pattern utterance. The role of semantics in filtering out candidates may be seen by considering the sequence.

PU: Is the chicken in the cooler?

FU: The potato salad?

Syntactically, "the potato salad" matches both "the chicken" and "the cooler". Semantically, it is 'closer' to "the chicken": they are both foods. Therefore, the ellipsis procedures should establish that as the candidate slot.

Semantic closeness in a system with a semantic network knowledge representation is determined from the element and superset hierarchy of the network. Given some collection of nodes N and a node m , the node n in N is most closely related to m if n and m belong to a common set (in the network) that does not include any other nodes of N . In network terms, node n is closest to m if, considering only element (e) and subset (s) arcs, n and m have the closest common ancestor. This closeness measure is a relative one. It can only be used to decide among a set of alternatives.

For each of the concepts in an utterance, a piece of semantic network is built. In particular, a noun phrase corresponds to a set of nodes and relations in the network. For each noun phrase, a single node in the network can be distinguished as central to the concept expressed in the noun phrase. The node corresponding to a definite NP is the node representing the object (or other concept) identified with the NP. (Intensional referents are an exception -- they may be treated like indefinites for the purposes of ellipsis.) For indefinite NPs, one node of the structure built for the NP is the 'head node'; the concept it

represents corresponds to the head noun of the noun phrase. Hence, corresponding to each candidate slot that passes through the syntactic constraints, there is a candidate node. The candidate node that is closest to the node corresponding to the EU is chosen as the matching node; the slot filled by its concept is the slot the EU is taken to fill.

To find the candidate node that shares the closest common ancestor with the EU-node, paths are grown by recursively following e and s arcs from each candidate node and the EU node. Having paths from two different starting nodes reach a common node indicates that the two nodes are elements of a common superset. If one of them is the EU node, the match has been found and the slot determined. Since all paths eventually reach the node UNIVERSAL (the top of the semantic net hierarchy), any path that reaches UNIVERSAL is eliminated from consideration. The paths traced for the sequence,

PU: Is the box-end wrench used to loosen the bolt?
EU: The socket wrench?

are shown in Figure X-1. Paths from the PU candidate nodes, which correspond to the DEFNPs, "the box-end wrench" and "the bolt" are shown with dashed lines. The path from the node corresponding to the EU is shown with a dotted line. The paths from 'W1' and 'W2' meet at 'wrenches'. It is important to note that the intermediate subsets, 'socket-wrenches' and 'box-end wrenches' are not needed for the algorithm to work (in fact, it works faster without them).

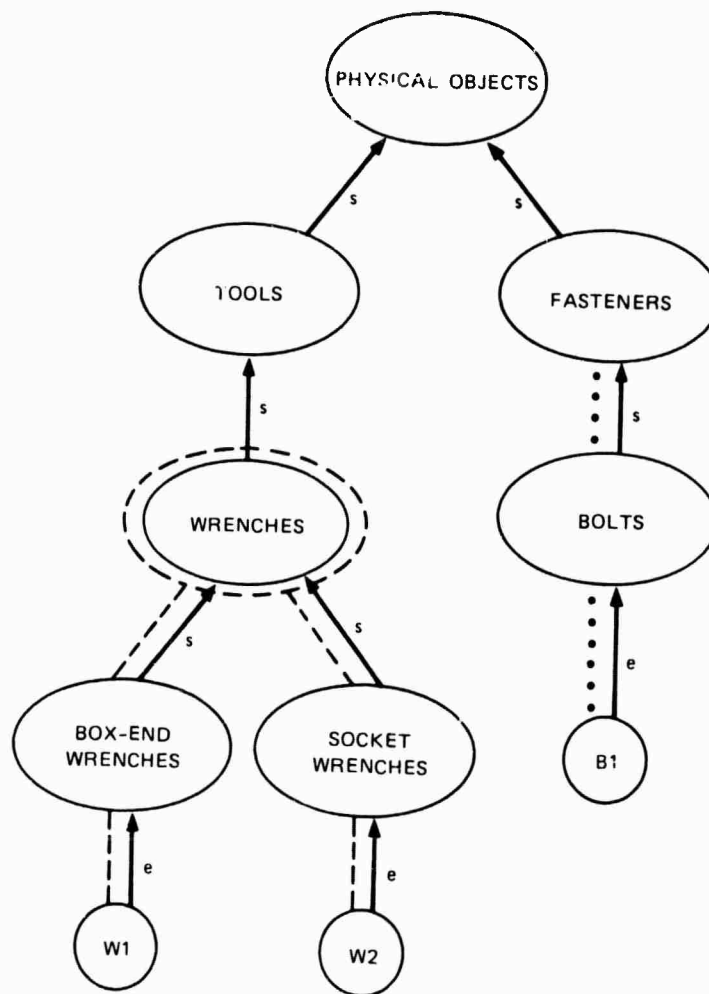


FIGURE X-1 PATH-GROWING ALGORITHM

Problems arise only when paths from two (or more) candidate nodes intersect with the path from the EU node at the same iteration of the algorithm. This can happen either because the paths all intersect (for the first time) at the same node, or because the paths from the candidates have intersected at some node and the path from that node intersects with the EU node's path. In either case the EU is ambiguous. Consider the following examples:

PU: Is the submarine faster than the carrier?
 EU: The patrol boat?

PU: Is the patrol boat faster than the carrier?
EU: The nuclear sub?

PU: Is the nuke faster than the diesel sub?
EU: The carrier?

PU: Is the nuclear sub faster than the patrol boat?
EU: The guided missile carrier?

PU: Is the patrol boat slower than the Ethan Allen?
EU: The guided missile carrier?

Since syntactic clues have already been used as a filter, discourse has no further way of disambiguating the utterance. A possibility, not explored in the current implementation, is to examine the syntactic roles of all of the candidates and see what keys to disambiguation might be asked of the speaker.

After a candidate is selected, there is also a need for semantic checking. This need is especially strong in a speech-understanding environment. Even though the phrase constituting an EU syntactically and semantically matches some phrase in the PU, it may not make sense semantically to substitute the EU for this phrase. For example, in

PU: Does Britain own a sub?
EU: A commander?

the EU matches the phrase "a sub" (they are both physical objects) but the substitution does not make sense (note that it would if the PU were, "Is there a sub in Naples?"). For this reason, a semantic check on the suitability of substituting the EU in the selected slot is always done. This check is in essence the same one that is done by the semantic composition routines when the original utterance (i.e., the PU) is

interpreted and the matching (slot) phrase is embedded in some higher level phrase. In building the PU, the semantic routines check the suitability of this embedding. In the above example, the phrase "own a sub" is checked. Before trying to substitute an EU, the discourse routines perform the same check with the EU. In the example, the plausibility of "own a commander" is checked and rejected. Such possibilities must be provided for in a system with speech input, since the acoustic routines may confuse "a commander" with "a Carpenter" (the name of a ship).

C. COMPLETING THE UTTERANCE

Completion of the elliptical utterance entails fitting it into the slot in the pattern utterance selected by the slot determination phase of the process. Semantic checks already have ensured that it is reasonable to substitute the EU for the NP that occupies the slot in the PU. The remaining step is to build a new structure using pieces of the PU and the EU. The use of a network partition to reflect the parse structure for an utterance is crucial to limiting the computing done in this expansion.

Elliptical expansion in an earlier version of the speech understanding system (see Walker et al., 1975) depended on having available a representation of the semantic interpretation of the complete PU in terms of the semantic representation of each of its constituents. The utterance expansion routines built a new net around

the semantic representation of the EU using all of the information from the semantic interpretation of the PU not superseded by information in the EU. But, in a speech system environment, interpretations of utterances are built up from partial interpretations. Each partial interpretation has been processed by both semantics and discourse to allow assignment of scores for determining which of the competing interpretations to work on next. As a result, the final semantic interpretation of an utterance is a combination of semantic representations of some constituents, discourse representations of other constituents, and semantic processing to handle quantification. The simple surgery of the original system no longer works because there is no complete semantic template available. For example, when a definite noun phrase is resolved, the node identified with the resolution, rather than the original semantic interpretation, is used in building representations for higher (embedding) phrases.

It would be possible for the semantic component of the system to build dual representations, one using semantics and one using discourse results, each time phrases were merged to make a higher level phrase. This duplication would make available a final semantic interpretation built only from semantic constituents. However, this solution would double the most expensive work done by semantics in building an interpretation. This doubling of effort would have to be done for all phrases that include NPs, even the false attempts that were not part of the final interpretation. Furthermore, such an expansion algorithm

requires copying all portions of the PU being used with the EU. In contrast, the algorithm described in this section overcomes both of these problems: it works using the combination of semantic and discourse representations, and it copies only those portions of an utterance that embed the slot filled by the EU.

To illustrate the basic algorithm we will consider the sequence

PU: What is the speed of the submarine?

EU: The carrier?

Figure X-2 shows the final semantic interpretation of the PU along with the semantic interpretation for each of the constituent phrases and the discourse interpretation of the NPs. The semantic representation of each constituent is in a separate space in the network. (The hierarchy of these spaces, shown by the heavy arrows, directly mirrors the parse structure of the utterance.)

As soon as the NP "the submarine" is encountered and semantics has built an interpretation for it, discourse is called. The submarine Churchill is found in focus (Chapter IX, Section B) and hence identified as the object referred to by the NP. Note that the node for the particular ship is used in the higher level (embedding) NP "the speed of the Churchill". Similarly, once the semantic interpretation for this NP is built, discourse is called and determines the node corresponding to the speed of the Churchill (which may or may not exist explicitly in the net; see Chapter IX, Section C). This node is then used in building the semantics for the whole utterance.

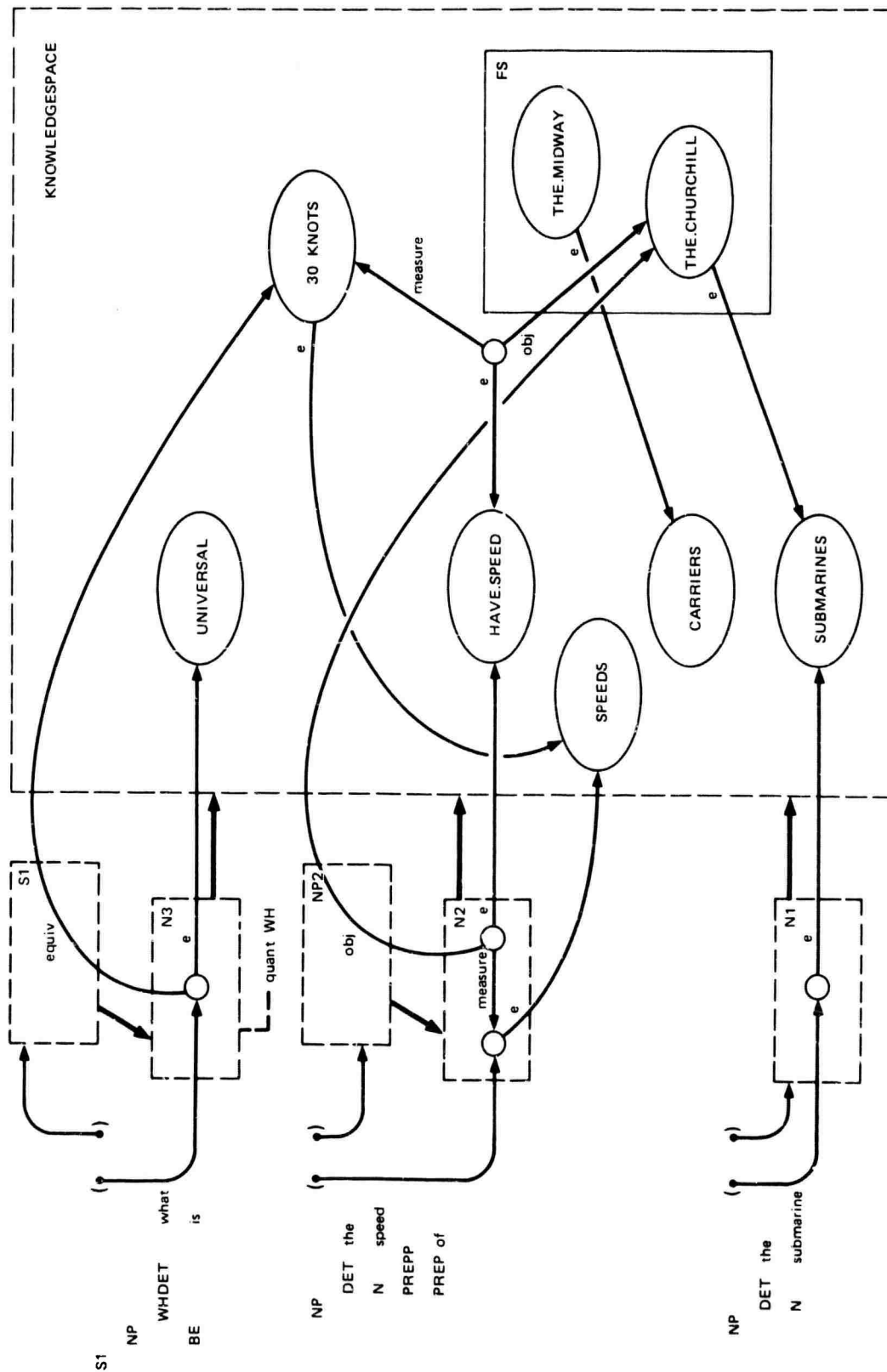


FIGURE X-2 REPRESENTATIONS FOR "WHAT IS THE SPEED OF THE SUBMARINE?"

Now consider what happens when the EU is encountered. The match of "the carrier" with the slot filled by "the submarine" is found as described in the preceding section. But the node for the Churchill is nowhere to be found in the utterance level semantics, which consists solely of the nodes and arcs in the vista of Spaces S1 and N3 of Figure X-2 (and of the knowledgespace nodes touched by those arcs). However, it is easy to find how any node was used in building a final interpretation of an utterance if enough information from the parse of that utterance is kept.

After an utterance is accepted, the discourse routines collect information about each of the NPs and VPs in history lists. In particular for NPs, the semantic interpretation, the discourse interpretation (which in some cases is identical to the semantic interpretation but is always different for definite NPs), the phrase of which the NP is a constituent (or in which it is "embedded"), and syntactic factors such as number and determination are noted in a table. For VPs, only the semantic interpretation and the embedding phrase need to be collected.

When an EU is encountered and the candidate slot found, the embedding phrase for the EU can be constructed from the embedding phrase for the phrase filling the slot in the PU. In the example, the embedding phrase for "the carrier" is NP2. The first step of substituting the EU in the slot is to copy the space(s) created when the embedding phrase was formed from its constituents and to substitute arcs

to the EU node for any arcs to the corresponding PU node. In the example, a new space NP3 corresponding to NP2 must be built with an arc to the Midway instead of the Churchill, as shown in Figure X-3. Note that it is not necessary to copy any of the structure built for other constituents of the embedding phrase. Network partitioning, in particular the visibility restrictions it imposes, enables each of these constituents to be viewed from the perspective of the new space. The result of this step is a new constituent for some higher level embedding phrase. Again the embedding phrase can be determined easily from the history lists. The process continues recursively until the embedding phrase is the utterance. Resolution of definite noun phrases (in particular, relational NPs) is performed, if relevant, when the new constituent is built. In the example, NP3 is built as shown in Figure X-3. Because this is a relational NP, it is passed to the resolution routines and the actual "speed of the Midway" is computed. Finally, this node is embedded in a copy of the utterance level semantics as shown in Figure X-4.

Notice that the visibility restrictions of network partitioning enable restricting the copying of constituents of the PU to those phrases embedding the slot filled by the EU. Looked at another way, only those phrases on the path from the slot to the root of the parse tree were copied. This attribute of the procedure may be seen even more clearly by considering the sequence

PU: Does Britain own the carrier?
EU: The U.S.?

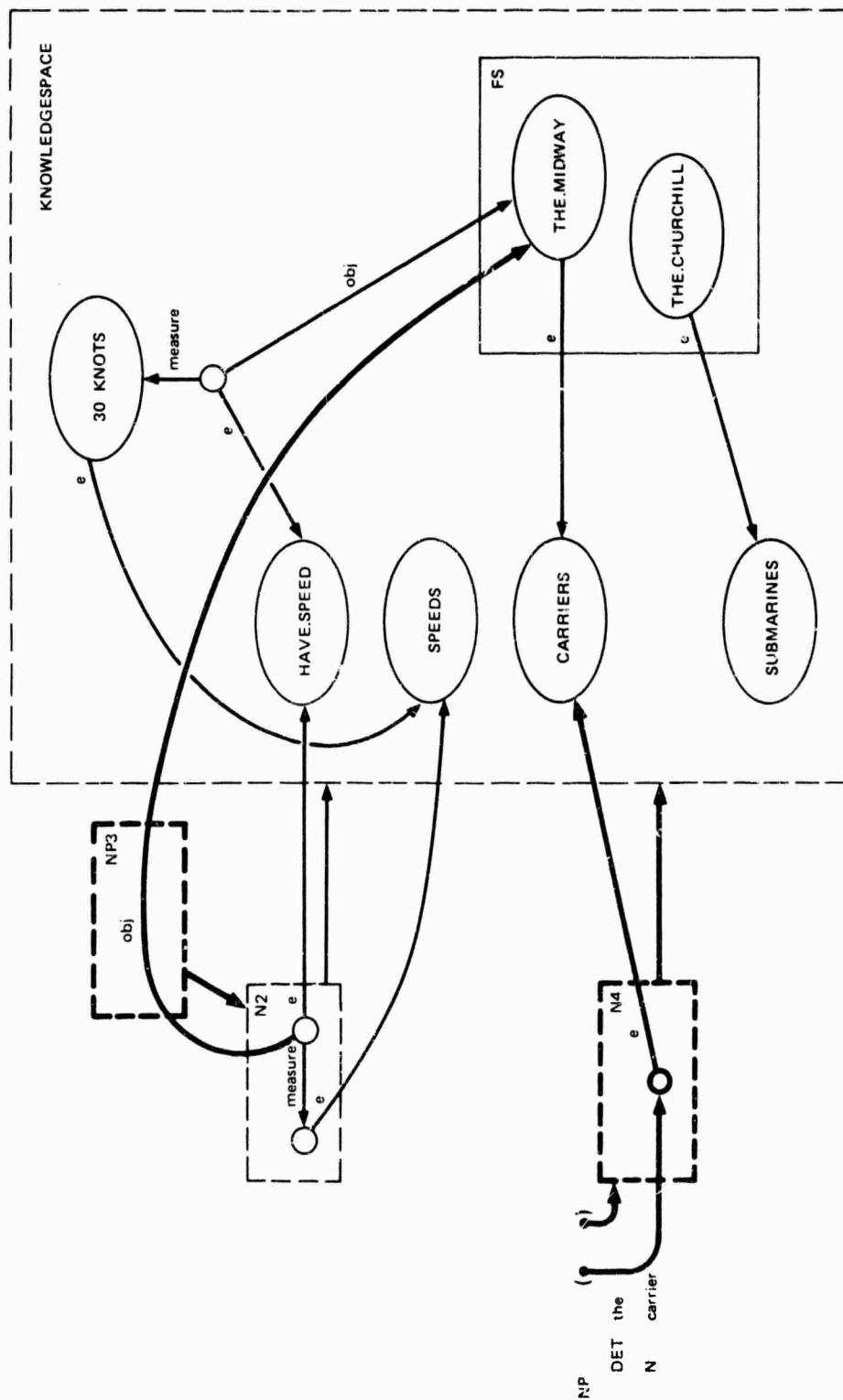


FIGURE X-3 RESOLVING "THE CARRIER" AND FIRST LEVEL EXPANSION OF ELLIPSIS TO "THE SPEED OF THE CARRIER"

and examining Figure X-5 and Figure X-6. The phrase "the U.S." corresponds to "Britain", a top-level constituent of the sentence. Only the space S1 and the 'agent' arc need to be copied in building the interpretation of the EU.

In the two examples presented so far, the EU is a definite NP. The only difference in handling indefinite NPs is that the head node (and other nodes and arcs) of the NP lie on spaces below the KNOWLEDGESPACE and these spaces must be copied in the first step of the substitution. Again, network partitioning minimizes the work; the whole collection of spaces for the EU becomes visible (and the spaces for the NP that fill the slot in the PU become invisible) when the new space is created for the embedding phrase.

A final problem occurs with quantified NPs. Consider the sequence:

PU: Does Britain own all of the subs?
EU: The carriers?

The quantifier "all" operates on the NP "the carriers" in the most natural interpretation of the EU. To obtain this result, some record must be kept of what quantifier, if any, applies to a phrase. But this is exactly what the semantic component does in the first step of handling quantifiers. When the NP "all of the subs" is constructed, the only thing that happens is the recording of the quantifier "all" on a space in the network of the NP. The actual rearrangement of the structure into one that corresponds to the network encoding of quantified statements (see Chapter V, Section E.2) does not occur until

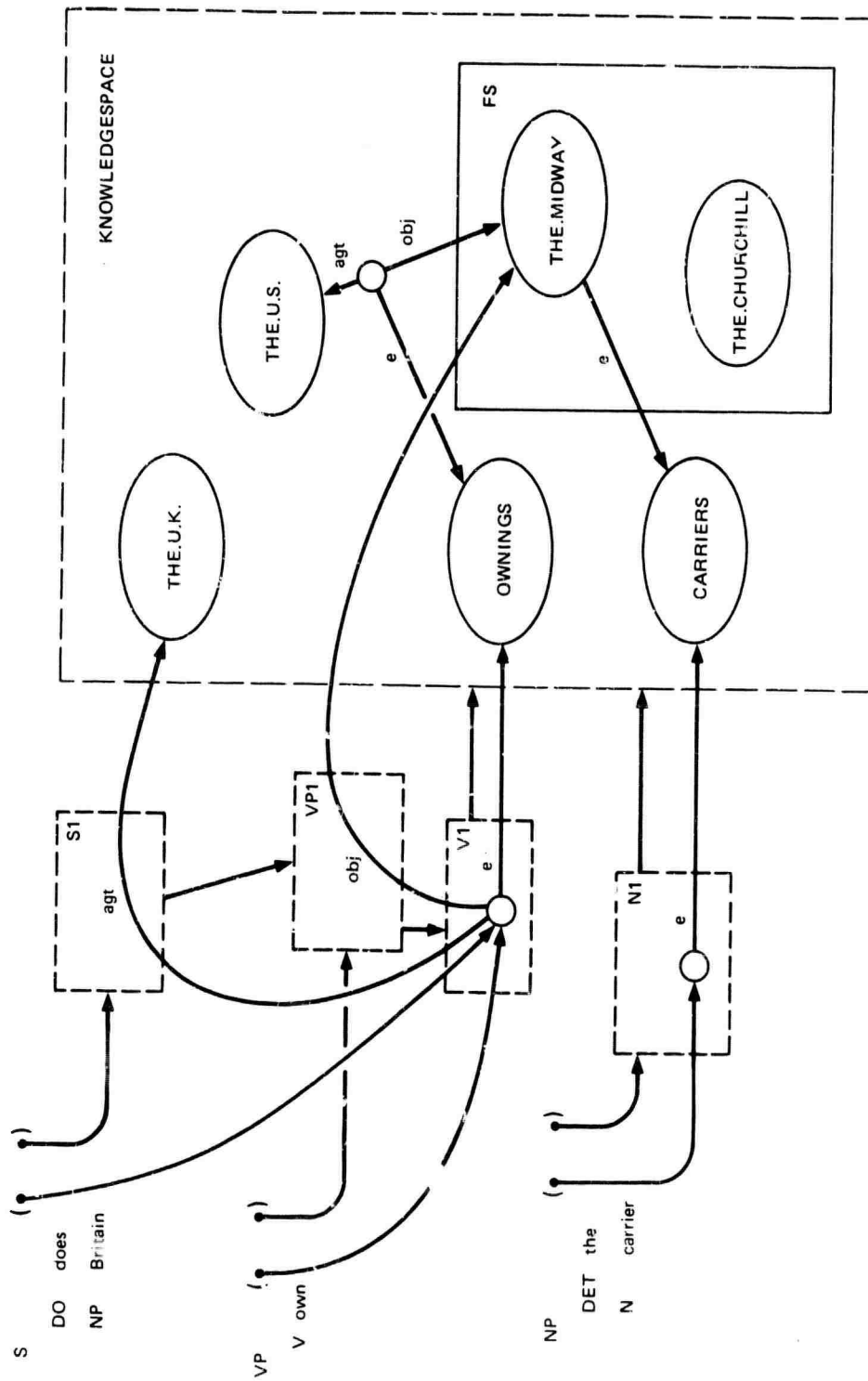


FIGURE X-5 REPRESENTATIONS FOR "DOES BRITAIN OWN THE CARRIER?"

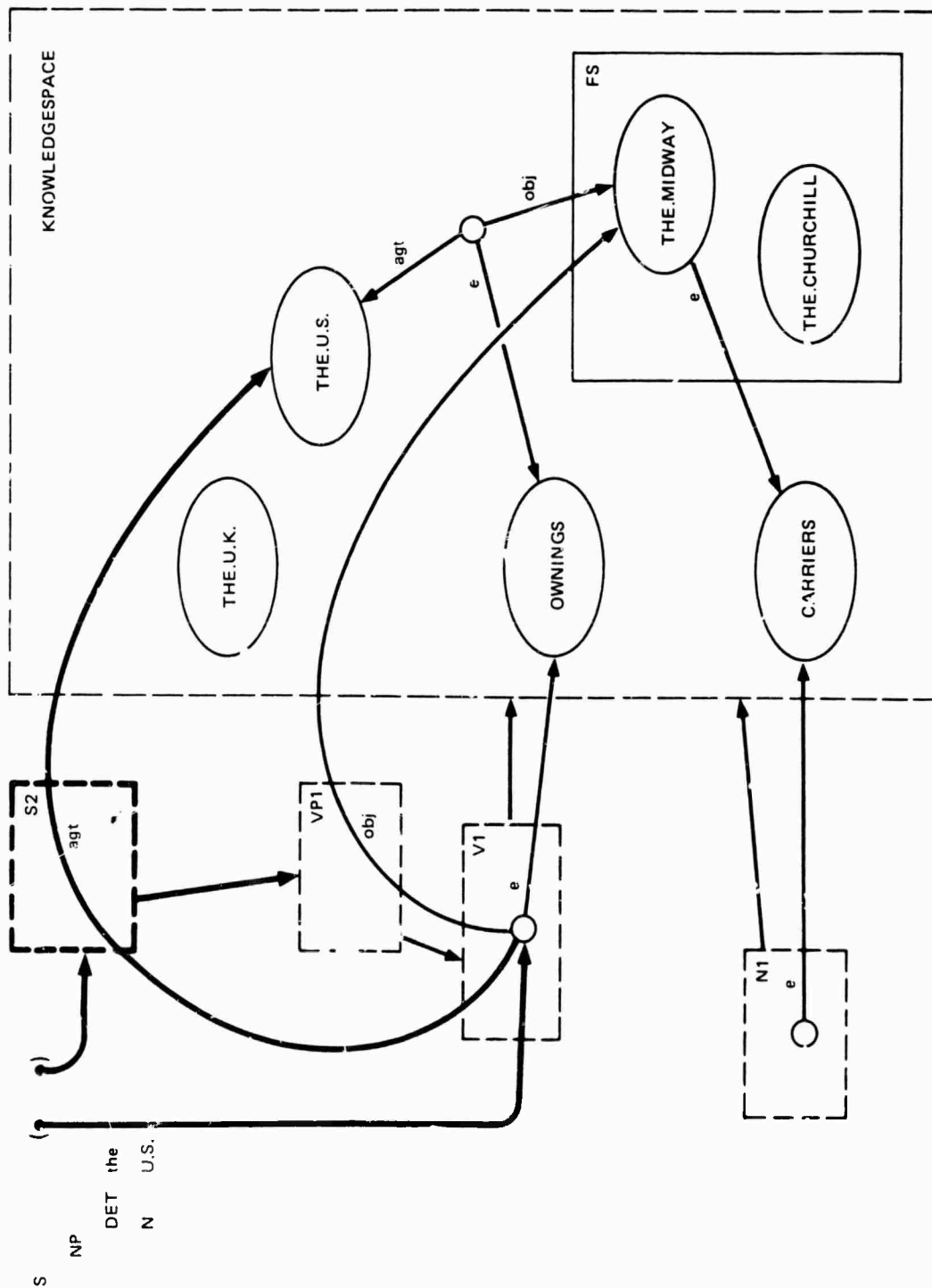


FIGURE X-6 EXPANSION OF THE ELLIPTICAL UTTERANCE, "THE U.S.?"

after the entire utterance is processed. Semantic processing must operate this way to capture the proper scoping of quantifiers. Discourse uses the tracks left at the parse structure level to transfer relevant quantifiers to elliptical utterances. In the sequence

PU: Does Britain own both carriers?
EU: Either carrier?

the EU is already quantified and the expansion process does not transfer the quantifier from the PU. The two-step process for handling quantifiers also means that an elliptical utterance when expanded may have different scoping than the PU. This difference in scoping occurs in the sequence

PU: Who built all ballistic missile submarines?
EU: Each Nuke?

The PU asks for the single builder of all ballistic missile submarines. The scope of "all" is inside of the scope of "who". In the EU, the scope of the "who" moves inside the scope of the quantifier, "each". For each nuke, the particular builder of that ship must be identified.

D. ELLIPTICAL RELATIONAL NOUN PHRASES

The ellipses discussed so far have all been structural in the sense that some syntactic pieces of an utterance have been left out; the structure of the utterance is incomplete. As a result, syntactic clues may be used to detect the ellipsis and to guide interpretation of it. The data base dialogs also contain elliptical utterances for which there are no syntactic clues. Consider the utterance: "What is the length?"; the ellipsis here is semantic. The utterance is syntactically, but not semantically, complete. "The length" is a well-formed NP; however, semantically, "length" assumes some object for which length is a relevant measure and implicitly conveys the relation of "having a length". The combination of this 'relational' attribute and definiteness indicates the need for an object. (If the utterance had been "what is a length", then no object would be required. The use of the indefinite determiner distinguishes this case.)

In essence, the verb like characteristics of the relational nouns cause a situation in which a phrase that appears to be syntactically complete is not. The object of the 'verb' is missing, but, since the verb is expressed through a noun, no syntactic indications of incompleteness occur. Case information appearing with the semantics of relational nouns can be used to detect this kind of ellipsis.

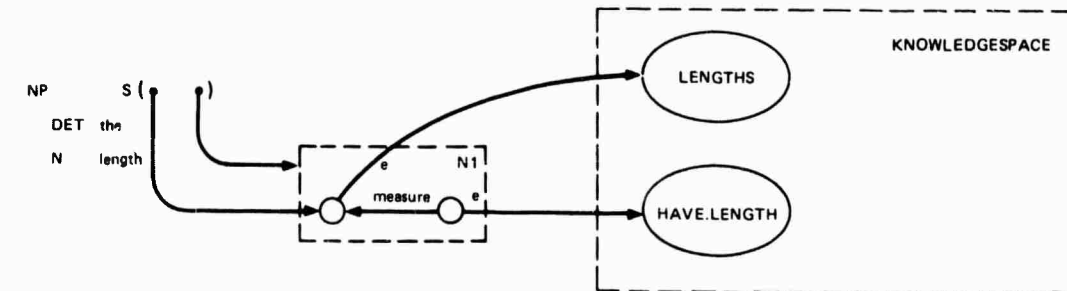
When a definitely determined relational NP (RELNP) is encountered, the discourse routines first check to see if all of the cases required

by the RELNP are present. If any are missing, ellipsis handling is invoked. The preceding utterance is examined to find objects for the empty slots. The procedure for finding candidate slots in the case of structural ellipsis can be used to determine which object in the PU best fills the missing case slot. Expansion of the elliptical RELNP is straightforward: a new space is created below the space for the RELNP and the space(s) containing the slot filler(s), and the case arcs are added to this space.

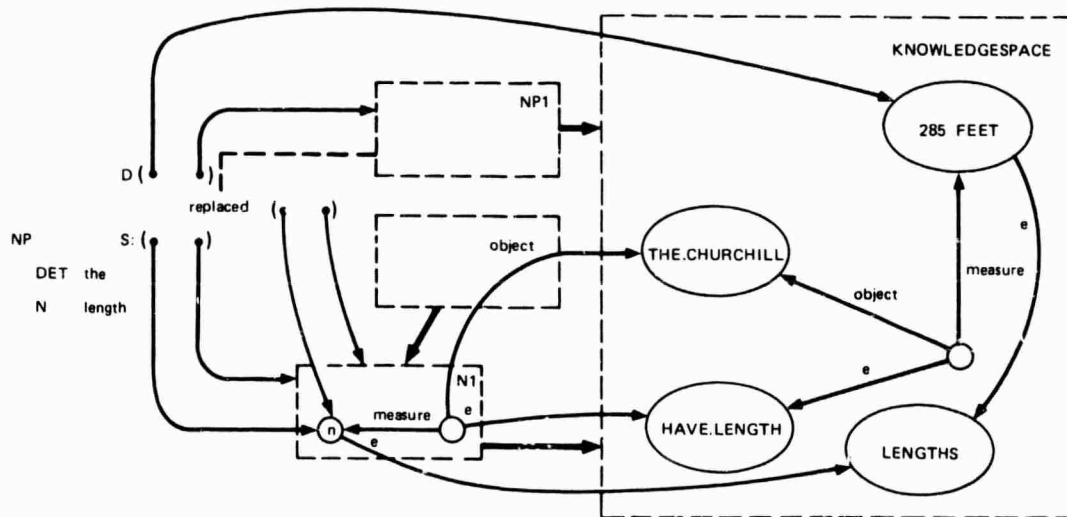
A compound case of structural and RELNP ellipsis occurs in the sequence

PU: What is the draft of the submarine?
EU: The length?

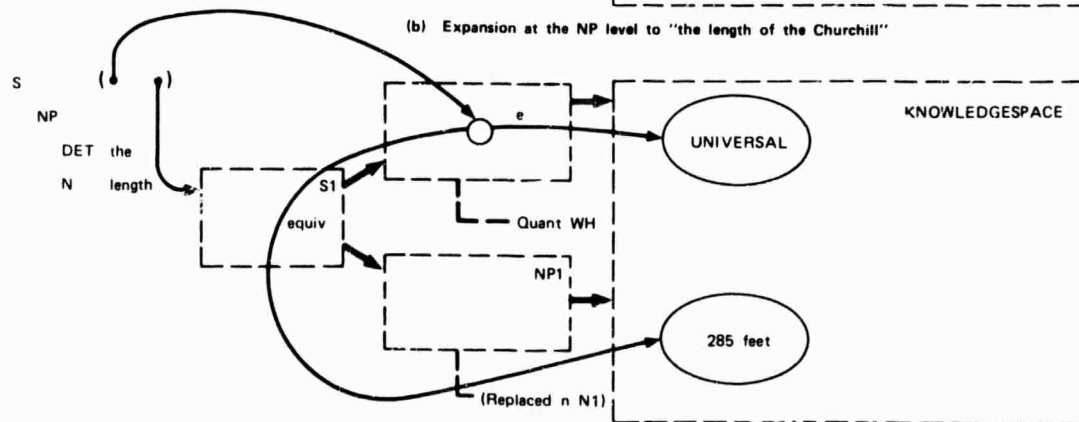
In processing this EU, the RELNP ellipsis is handled at the noun phrase level resulting in the structure of (a) in Figure X-7 being transformed into the structure of (b). The structural ellipsis is handled at the utterance level. At this point the problem is equivalent to processing the EU, "the length of the submarine". The result appears in (c) of Figure X-7.



(a) Semantic interpretation of the NP "the length"



(b) Expansion at the NP level to "the length of the Churchill"



(c) Expansion to full utterance

FIGURE X-7 EXPANSION OF THE ELLIPTICAL UTTERANCE, "THE LENGTH"

E. LIMITATIONS AND EXTENSIONS

The ellipsis-handling capabilities described in this section are limited in at least two ways. First, the slot-determining procedures depend on the presence of a matching phrase in the pattern utterance. However, an elliptical utterance may be a modifying phrase to be added to the PU; in this case, there will be no matching phrase, but rather a missing (optional) constituent. Second, we restricted our treatment to isolated noun phrases (and nominals) serving as utterances and semantically elliptical RELNPs, both because we wanted to reduce the number of competing hypotheses that would have to be considered in the interpretation of a spoken utterance and because no other instances occurred in the dialogs. However, the algorithm for expanding an elliptical utterance is general. In the remainder of this section we discuss these limitations and present the extensions necessary for handling less restricted forms of ellipsis.

The major limitation of the current ellipsis routines stems from the assumption that the EU will fill a single slot in the PU, which is not true of ellipsis in general. At the utterance level, the general case is that any number of constituents may be present or missing in the EU. In the sequence,

PU: Did you take the coat to the cleaners?

EU: The shoes to the shoemaker?

the EU contains an object NP and an adverbial prepositional phrase. The subject NP and the VP must be retrieved from the PU. This kind of

ellipsis is even more common when more complex sentences are considered. In particular, when two clauses or phrases are conjoined, the second is often elliptical; consider the above sequence joined by "and". Rather than looking for a single slot filled by the EU, the ellipsis routines should determine the constituents missing from the PU and then build the full utterance (the latter step would be quite similar to the work done by the semantic composition routines).

The mechanism for handling ellipsis this way would entail a closer coupling of syntax and discourse and would proceed basically as follows. The parsing routines would determine which constituents of the utterance were present in the EU and which were missing, on the basis of the context-free structural description associated with each rule in the language definition. Using this information and the parse of the PU, the discourse routines would build the complete utterance in a manner similar to the one now used for expansion. The only difference would be that several components might get replaced at once. Both semantic and syntactic checking could be done based on the mapping between the structure of the PU and that of the completed EU.

Adopting such a strategy eliminates two major limitations of the current approach. First, the EU may consist of any number of constituents, not just a single NP (the only exception to this restriction in the current routines is with RELNP ellipsis). In particular, the EU may consist solely of a modifying phrase not present

in the PU, as in the sequence,*

PU: Plot the distribution of soybeans.

EU: In the year 2000.

Second, the extension to handling NP and VP ellipsis is straightforward. The only additional step needed is to determine the NP (or VP) in the PU that matches the elliptical phrase. The PU phrase then takes the role of the PU and the elliptical phrase takes the role of the EU in the above description. The result of the processing is a complete NP (or VP) to be used in building the rest of the utterance. For example, in the sequence,

PU: Is the Churchill the smallest sub?

EU: Is the Lafayette the largest?

the elliptical NP "the largest" gets matched with the PU phrase "the smallest sub", and is then expanded to "the largest sub". This complete NP can then be used in the (now complete) EU.

Processing the kinds of ellipsis occurring in the question answering pairs of the task dialogs also entails only one additional step. The question (PU) must be transformed before it can be used as a template. As an example, consider the sequence

PU: Which bolts did you tighten?

EU: The front bolts.

The PU must get transformed to "You did tighten which bolts", then an I/you transformation must be done. Then the EU can be placed in the slot (nicely indicated by the WH-phrase). A means of expanding the

* Thanks to W. H. Paxton for this example and for a suggestion of how to handle it. The content of this section was greatly influenced by discussions with him.

language definition to facilitate this kind of processing is currently being explored.

XI RESPONDING ON THE BASIS OF THE SEMANTIC TRANSLATION

Prepared by Gary G. Hendrix

CONTENTS:

- A. Perspective
- B. Interactions with the Deduction Component and the English Generator
 - 1. Yes/No Queries
 - 2. WH Queries

A. PERSPECTIVE

Once a semantic translation has been constructed for an utterance, a language understanding system will respond in accordance with the nature of its interpretation of the input and within its abilities to perform tasks. The range of sophistication in response ability is potentially quite broad. At one extreme, a system might be able to build literal interpretations of inputs but be completely unable to act upon them. At the other extreme, a system might record who uttered the input; consider what the meaning of the input is in terms of the system's perspective on the knowledge, beliefs, goals, and social behavior of the speaker; and then consider how to act upon the input in such a way as to maximize the system's own potential for reward, as determined by its own value system, goals, aspirations, and predictions concerning how its actions will affect future states of the world.

Possible actions to be performed by a sophisticated system would include updating its world model to reflect new input information; supplementing the model by making inferences and deductions; both finding answers logically or even performing physical experiments to determine them; taking physical action in response to commands and requests; and planning and executing sequences of actions to achieve or maintain goals in the face of new input data.

The response component of the SRI speech understanding system is rather limited in its scope, reflecting the project's emphasis on an intelligent interpretation of the utterance as opposed to subsequent processing. The resources that may be marshalled by the responder include a component that performs logical deduction (see Chapter XII), a natural language generator that converts network structures into appropriate English expressions (see Chapter XIII), and a routine for drawing partitioned network structures.* No resources for permanently augmenting the task domain model, doing planning, reasoning about goals and beliefs, or performing motor activities are available.

Given its current set of resources, the task of the responder is to determine which inputs are requests for information that may be acted upon by the deduction component and which are not. For those that are not, a representation of the corresponding partitioned network structure is drawn to express the system's interpretation of the utterance. For

* This routine was programmed by Par Emanuelson of Linköping University in Sweden while he was a visiting research engineer at SRI. It is illustrated in the example presented in Chapter I, Section C.

those inputs that are suitable for deductive processing, the responder formats a call to deduction and interprets the results returned by it. Depending upon the type of information requested and the results returned, the responder will either produce a specified response, like YES or NO, or will invoke the English generator to express the results of the deduction processing.

B. INTERACTIONS WITH THE DEDUCTION COMPONENT AND THE ENGLISH GENERATOR

The input to the response component is a space T that represents the interpretation of the spoken utterance. Such translation spaces are produced by the quantification phase of the semantic translation process. To determine what action is to be taken in response to an input, the responder examines space T, looking for either the structure of Figure XI-1 or the structure of Figure XI-2. For question answering to be performed, space T must contain exactly the structures shown and contain no additional structures. The special sets REQUESTS.YN and REQUESTS.WH are used in the encoding of YES/NO and WH questions, respectively, and are discussed in Chapter V, Section E 3.

The structures of Figure XI-1 and Figure XI-2 are not necessarily the interpretations of questions. For example, the command

"Give me the speed of the Henry.L.Stimson."

translates into the same request for information that would be produced for

"What is the speed of the Henry.L.Stimson?"

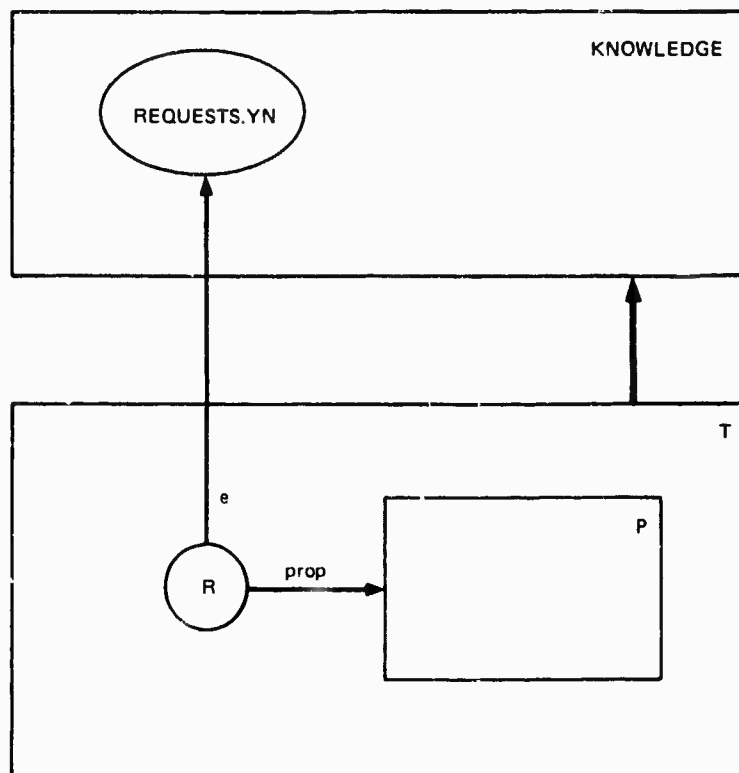


FIGURE XI-1 SCHEMATIC OF YES/NO QUESTION

On the other hand, not all queries fall into one of these two forms. For example, the query

"Who built each destroyer?"

which is interpreted as

"For each destroyer, who was its builder?"

does not fall in either category but rather embeds the request structure of Figure XI-2 in a universally quantified expression. Some questions involving quantifiers are accepted. For example,

"Did General.Dynamics build all of the Lafayettes?"

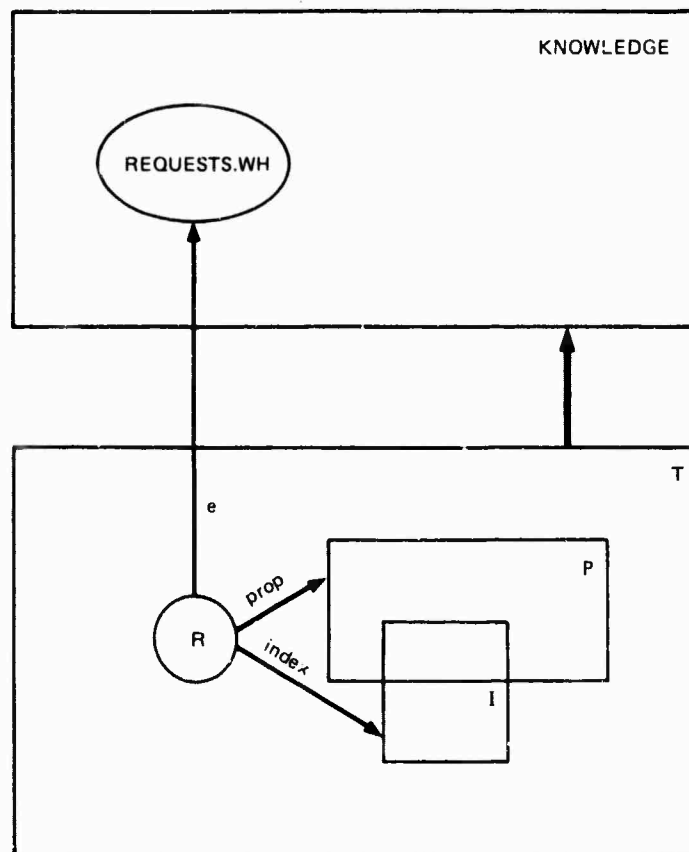


FIGURE XI-2 SCHEMATIC OF WH QUESTION

follows the form of Figure XI-1.

The system's understanding of any utterance that does not follow one of the forms cited above is expressed by printing out a drawing of the network of the utterance's translation space *T* (and the structures of spaces embedded as supernodes in *T*, and so on).

1. YES/NO QUERIES

For top-level YES/NO questions, which follow the form of Figure XI-1, the responder calls the deduction component (see the discussion in Chapter XII for an explanation of its input/output characteristics) with a QVISTA and a KVISTA. The QVISTA used in this call is a one item list containing only space P, and the KVISTA is a one-item list containing only space KNOWLEDGE of the system's task domain model.*

Essentially, space P (and its embedded spaces if it contains logical connectives such as IMPLICATIONS, NEGATIONS, or DISJUNCTIONS) represents a proposition upon whose truth rests the answer to the YES/NO query. The job of deduction is to test the truth of P against the domain model encoded in the KNOWLEDGE space. Conceptually, this is done by pattern matching the structures of P against the structures of KNOWLEDGE.

If the proposition can be proved true, then deduction returns an association list describing how structures in P may be instantiated by structures in KNOWLEDGE. If the proposition can be proved false, deduction provides a counterexample (that is, an instantiation in a negation space). In addition to associations between P structures and

* KVISTA is the orthodox vista of KNOWLEDGE, but QVISTA is created by the responder. The orthodox vista of P is the list (P T KNOWLEDGE). In general, the QVISTA and KVISTA supplied to deduction may contain an arbitrary number of spaces. In fact, calls to deduction from discourse typically use QVISTAS containing multiple spaces created by the semantic composition routines.

KNOWLEDGE structures, the association list contains a pair of the form (ANSWER . value). For YES/NO questions, the responder returns this value to the speech understanding system executive for printing. The value will be YES, or NO, or, in those cases where deduction can neither prove nor disprove the proposition, UNKNOWN.

2. WH QUERIES

Top-level WH questions have a prop space P just like YES/NO questions. The responder uses this space to set up a call to deduction that is identical to the call for YES/NO questions. If the ANSWER value on the returned association list is NO, then the responder reports that the underlying proposition was faulty. For example, the report "NO SUCH PERSON OR THING" would be produced for the queries

"What submarine is a destroyer?"

and (more convincingly)

"Which destroyers are nukes?"

(There are 3 nuclear destroyers.) If users are not expected to ask WH questions with false propositions, then the language definition may include a proviso that if this condition arises, the executive's score of the query will be lowered and parsing resumed in search of a higher scoring interpretation of the input.

If the ANSWER value on the association list is UNKNOWN, then deduction could neither prove nor disprove the proposition of the question. The responder returns an appropriate message.

If the ANSWER value on the association list returned by deduction for a WH query is YES, then the association list contains a mapping from the query's proposition onto one of the proposition's (possibly many) instantiations. This map holds the (an) answer to the query. For example, upon receiving the translation of the query

"Who built the Henry.L.Stimson?"

which is shown in Figure XI-3, the responder calls deduction with a QVISTA of (P). Then deduction returns an association list whose form is approximately as follows:

```
[(ANSWER . YES)
 (VISTA . (KEXTENSION KNOWLEDGE))
 (X . General.Dynamics)
 (Y . DERIVED.031)
 (<e X LEGAL.PERSONS> . <e General.Dynamics LEGAL.PERSONS>)
 (<e Y BUILDINGS> . <e DERIVED.031 BUILDINGS>)
 (<agt Y X> . <agt DERIVED.031 General.Dynamics>)
 (<obj Y Henry.L.Stimson> . <obj DERIVED.031 Henry.L.Stimson>)]
```

This association list shows the mapping between the original proposition P, which may be paraphrased as:

There is some legal person X,
and there is some building situation Y,
and the agt of Y is X,
and the obj of Y is the Henry.L.Stimson.

and its instantiation

General.Dynamics is a legal person,
and DERIVED.031 is a building situation,
and the agt of DERIVED.031 is General.Dynamics,
and the obj of DERIVED.031 is the Henry.L.Stimson.

In particular, 'X' is associated with 'General.Dynamics'. Since 'Y' is the node of the WH query's index space I, its image, 'General.Dynamics', is the answer to the question.

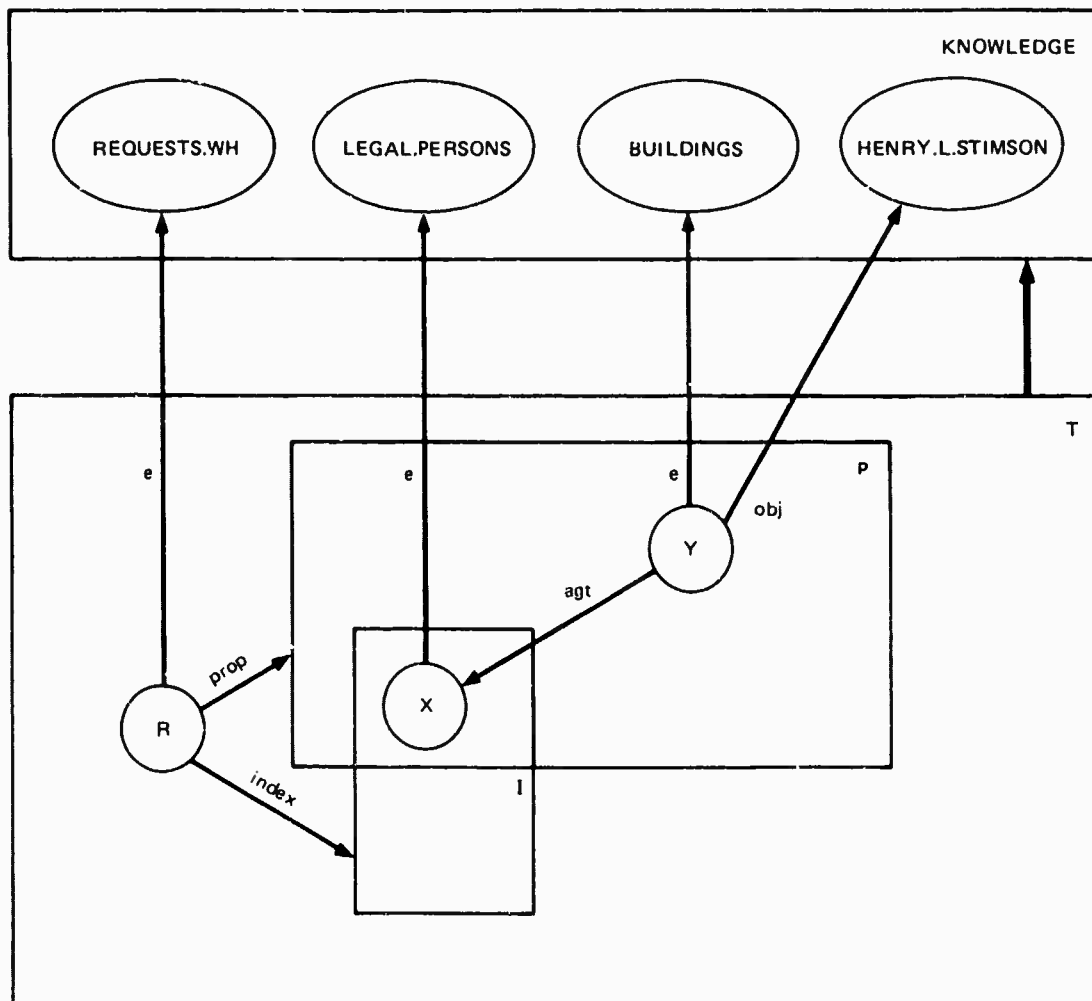


FIGURE XI-3 TRANSLATION OF "WHO BUILT THE HENRY.L.STIMSON?"

The way in which the responder expresses answers to WH queries depends upon the setting of system variable SENTENCEFLG and the nature of the query's index space. If SENTENCEFLG is NIL and the index space contains exactly one node, then the image of that node is passed to the English generator for translation into an English phrase. In the

example above, the node 'General.Dynamics' would be passed to the generator and the list (General Dynamics Corporation) would be returned. The vista that is the value of VISTA on the association list -- for the example, (KEXTENSION KNOWLEDGE) -- is passed as a second argument to the generator. This vista contains not only the KNOWLEDGE space, but other spaces that have been created by deduction to encode nodes and arcs (e.g., 'DERIVED.031') that were derived in the process of instantiating the proposition of space P. Since the answer to a WH question might be a derived node, this vista may be needed by the generator to provide a sufficient vantage from which to view the node. In the example, the instantiation of 'X' is not derived, but lies on the KNOWLEDGE space. (The instantiation of 'Y' is derived.)

If the SENTENCEFLG is set to TRUE or if multiple nodes lie on the index space I, then a complete sentence (or sequence of sentences) describing the instantiation of P in KNOWLEDGE (and its extensions) is generated as an answer to the query. To do this, the responder creates a new (scratch) space G below KNOWLEDGE and copies all the instantiations of the structures of P onto this space. That is, each network structure appearing on the right side of a pair on the association list returned by deduction is copied onto G. (This copying onto a new space is performed at a fraction of the cost of creating new structures.) The new space G partitions off the instantiations of P from other network structures. This space is then passed to the English generator, which is responsible for expressing all the concepts encoded

on the space. For the example query, that set of concepts may be expressed as

"The General Dynamics Corporation built the Henry.L.Stimson."

XII THE DEDUCTION COMPONENT

Prepared by Richard E. Fikes

CONTENTS:

- A. Introduction
- B. Element Parity
- C. The Environment Tree
- D. The Executive for the Deductive Component
- E. Generating Candidate Bindings for a Selected QVISTA Element
- F. Ramifications of a Proposed Binding
- G. The Binder
 - 1. QVISTA Top Level Elements
 - 2. QVISTA Disjunctions
 - 3. QVISTA Implications
 - a. Consequent Match
 - b. Antecedent Match
 - 4. QVISTA Negations
- H. Deriving Element-of and Subset Relations Using Taxonomies
- I. Simplification of Negations
- J. The KVISTA Extractor
 - 1. KVISTA Extraction Rules
 - a. KVISTA Disjunctions
 - b. KVISTA Implications
 - i. Consequent Match
 - ii. Antecedent Match
 - c. KVISTA Negations
 - 2. An Example
- K. The QVISTA Extractor
 - 1. QVISTA Extraction Rules
 - a. QVISTA Disjunctions
 - b. QVISTA Implications
 - i. Consequent Match
 - ii. Antecedent Match
 - c. QVISTA Negations
 - 2. An Example
- L. Procedural Augmentation
 - 1. E Arc with Bound Nodes
 - 2. Sets Defined in the QVISTA
 - 3. Applications and Keyed-Applications
 - 4. Efficiency Considerations
- M. Two Examples

A. INTRODUCTION

This chapter is a progress report on the deduction component, a facility for retrieving information from procedurally augmented, partitioned semantic networks. Several complete implementations have been constructed during the last two years, each incorporating major design changes from its predecessor, and our exploration of new design ideas continues. We describe here a set of facilities embodying the major design ideas that have evolved from our experience with these systems.*

When the response component determines that an utterance is a request for information, the deduction component is called to process the net structure corresponding to the utterance, relating it to other information stored in the domain model. The deduction component is capable of retrieving information explicitly stored in the nets, deriving information using general statements stored as theorems in the net, and calling user supplied functions pointed to in the net that obtain information from knowledge sources other than the net such as data files.

* Many people in the SRI Artificial Intelligence Center have contributed to the development of the deduction component. Gary Hendrix has been a major partner in the design effort throughout the project. Nils Nilsson has been an important contributor to the deductive machinery, and Jonathan Slocum both designed and implemented the data base interface. Mike Wilber and Rene Reboh were major participants in the overall implementation effort.

For example, if a user asks "Who built the Henry L. Stimson?" and the knowledge net contains the fact 'General Dynamics built the Henry L. Stimson', then the answer would be determined by simply retrieving that fact. If the knowledge net did not contain that fact but instead contained the theorem 'General Dynamics built all of the Lafayettes' and the fact 'The Henry L. Stimson is a Lafayette', then the same query would be answered by using that theorem and that fact to deduce the answer. Alternatively, if the knowledge net did not contain those facts or that theorem, but instead contained the theorem 'Function SHIPDATA can be called to determine the builder of any given ship' and the fact 'The Henry L. Stimson is a ship', then the same query would be answered by determining that the Henry L. Stimson is a ship and then calling the function SHIPDATA.

The deduction component accepts as input a vista called QVISTA containing the network translation of an English query and a vista called KVISTA containing the knowledge base from which answers to the query are to be retrieved. The QVISTA is the 'proposition' portion of the query (see Chapter V, Section E.3) and can be thought of as representing a 'pattern', with the QVISTA elements (i.e., arcs and nodes) being the pattern's variables. Processing entails seeking a 'match' in the KVISTA for the query pattern. A successful match produces a list containing a 'binding' for each QVISTA element to a corresponding KVISTA element. For example, if the query is "What submarines did General Dynamics build?", then the QVISTA would be as

shown in Figure XII-1. The KVISTA would be examined for elements of the BUILDINGS set that have an outgoing agt (agent) arc to GENERAL.DYNAMICS and that have an outgoing obj (object) arc to a node that has an outgoing e (element) or de (distinct elements) arc to SUBMARINES. The to-node of the obj arc of each such element of BUILDINGS represents an answer to the query.

The deduction component is a 'generator' of answers, each answer being in the form of a list containing a binding for each QVISTA element. After a bindings list is returned, it can be repeatedly 'pulsed' to find as many different answers to the query as desired. Hence, in the previous example, each time it is pulsed, it will indicate another submarine that General Dynamics built.

Included on the bindings list produced is an 'answer pair' whose first member is "ANSWER" and whose second member is either "YES", "NO", or "UNKNOWN". Each time a binding is found for each QVISTA element, the answer pair in the generated bindings list indicates a "Yes" answer. If it has been proved that no possible set of bindings exists for the elements in QVISTA, then the answer pair will indicate a "No" answer to the query. When it is not possible to find either another set of bindings for the QVISTA or any set of bindings for the negation of the QVISTA, then a one-element bindings list is generated, indicating an answer of "Unknown".

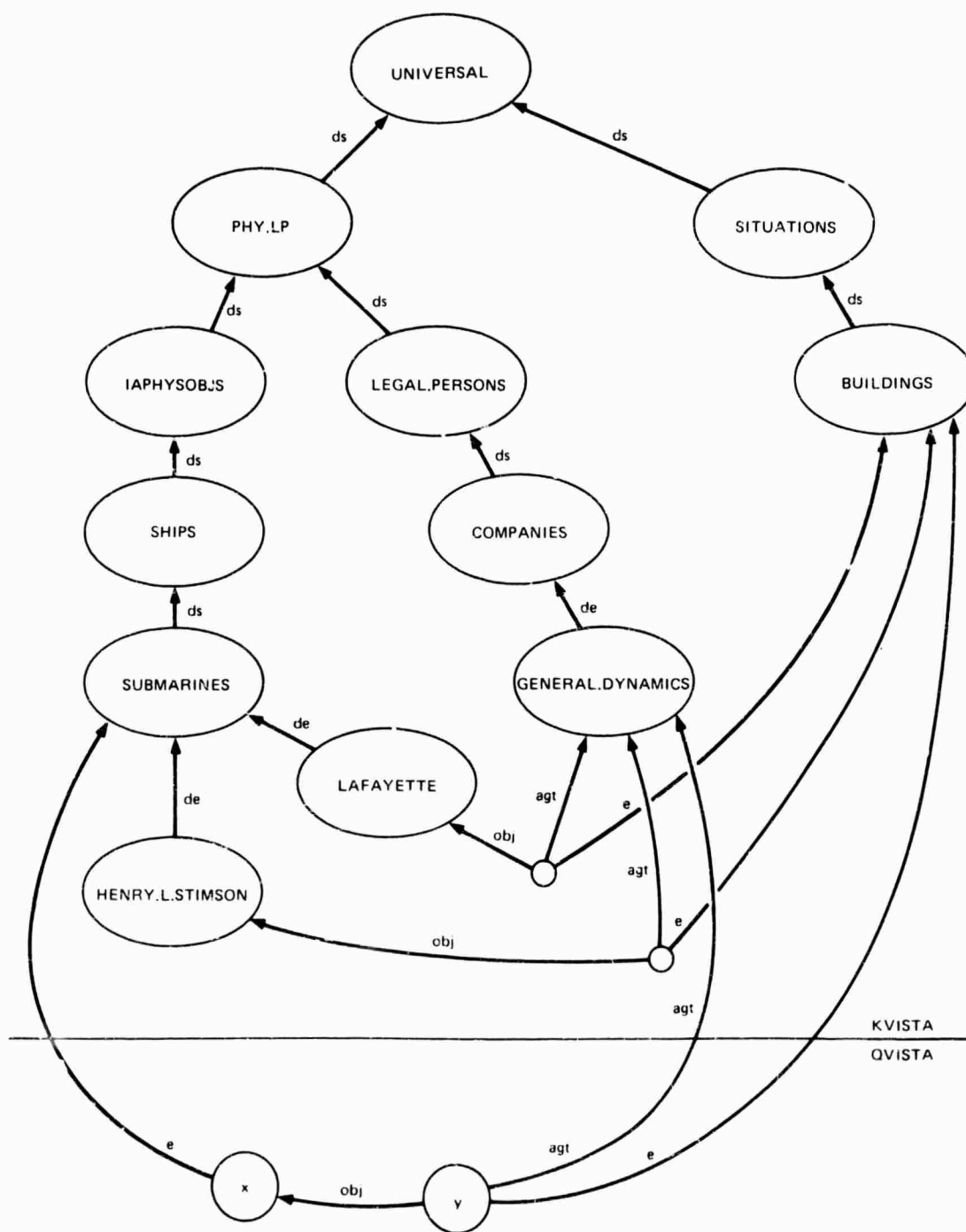


FIGURE XII-1 KVISTA AND QVISTA FOR THE EXAMPLE QUERY "WHAT SUBMARINES DID GENERAL.DYNAMICS BUILD?"

The deduction component retrieves information directly from the KVISTA using the indexing properties of the nets. It contains derivational machinery that provides the equivalent of a logically complete first-order predicate calculus theorem prover. It also contains procedural augmentation facilities for applying user-supplied semantics-based deduction functions. The following examples indicate how these capabilities contribute to the answering of queries.

Consider the KVISTA and QVISTA shown in Figure XII-2. The indexing properties of the net would be used to find the following bindings:

- node Y to node B,
- node X to node General.Dynamics,
- arc Y--obj-->Henry.L.Stimson to arc B--obj-->Henry.L.Stimson,
- arc Y--agt-->X to arc B--agt-->General.Dynamics, and
- arc Y--e-->BUILDINGS to arc B--e-->BUILDINGS.

A binding for arc X--e-->LEGAL.PERSONS must be derived since no e arc exists in the KVISTA between node General.Dynamics and node LEGAL.PERSONS. The required arc is easily derived by finding the 'chain' of de and d's arcs that connects the two nodes.

Another example is shown in Figure XII-3. To answer this query, it is necessary to carry out a derivation using the implication in the KVISTA that represents the statement "General Dynamics built all of the Lafayettes". The derivation will proceed by setting up a subproblem to find bindings for the instance of the implication's antecedent that requires the Henry L. Stimson to be a member of the set of Lafayettes. When that subproblem is solved, a new member of the BUILDINGS set will be added to the KVISTA that will provide the desired bindings.

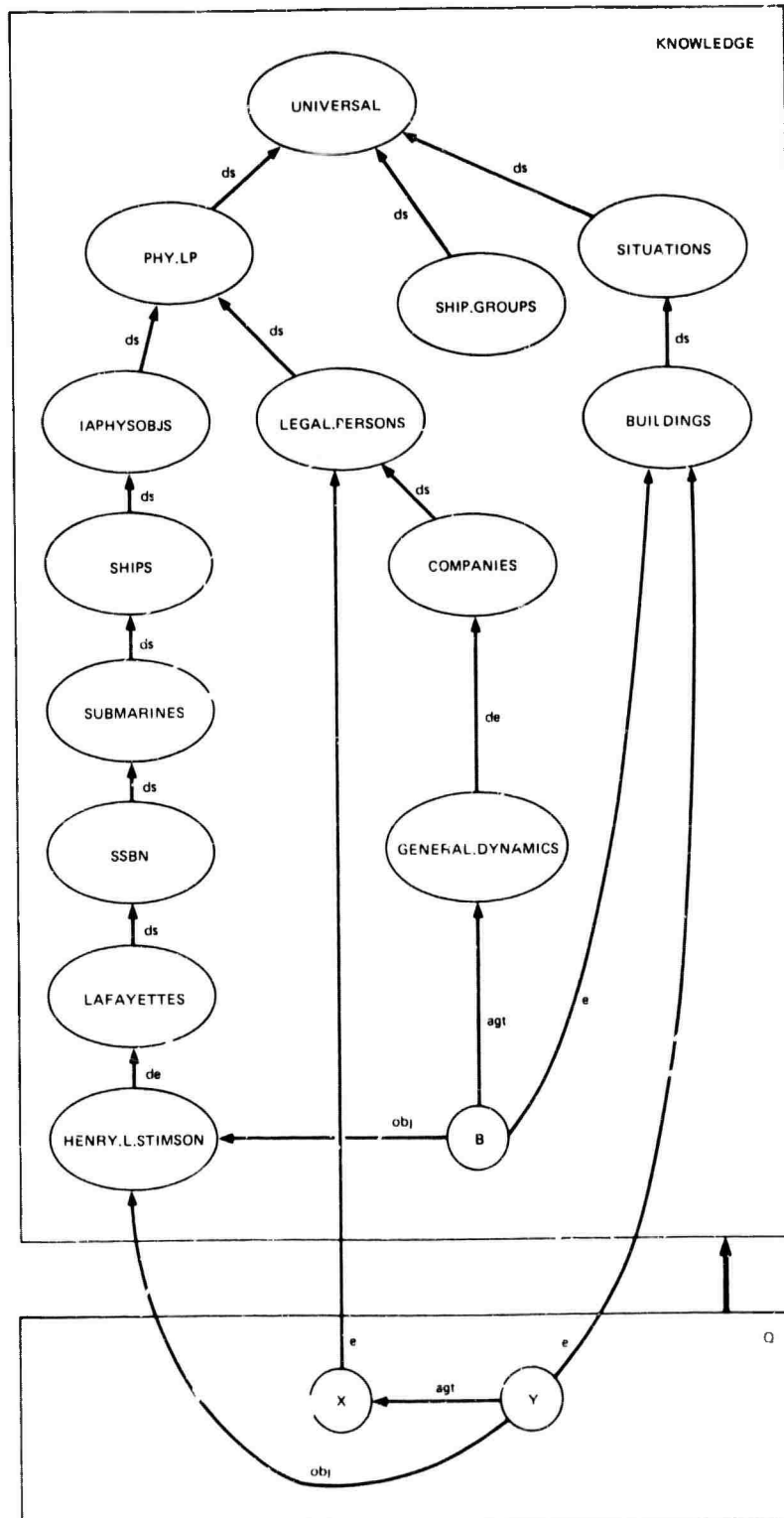


FIGURE XII-2 AN EXAMPLE QUERY, "WHO BUILT THE HENRY.L.STIMSON?", WHOSE ANSWER IS EXPLICITLY AVAILABLE

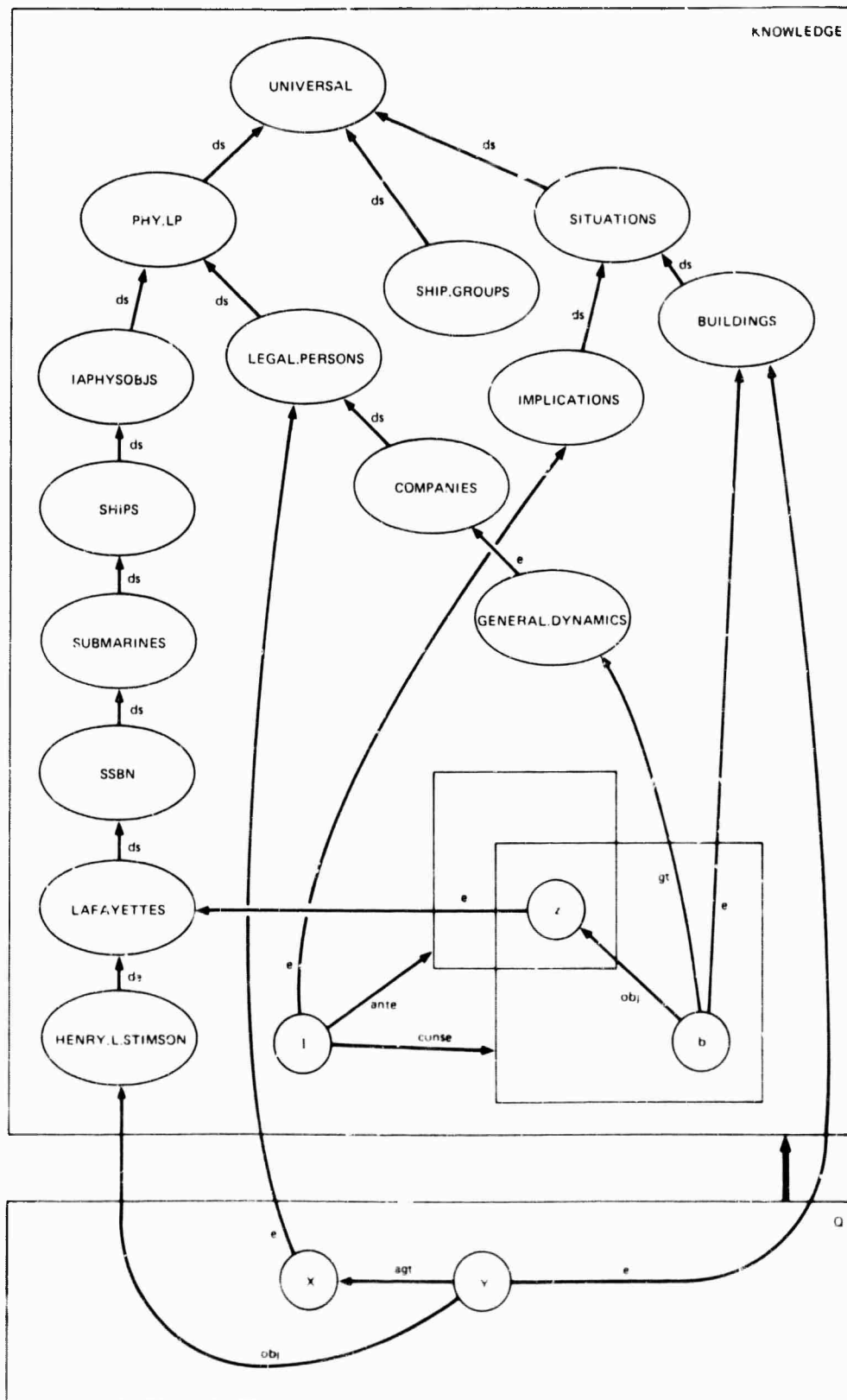


FIGURE XII-3 AN EXAMPLE QUERY, "WHO BUILT THE HENRY L.STIMSON?", WHOSE ANSWER IS INTERNALLY DERIVABLE

Figure XII-4 shows an example with a KVISTA that contains a theorem indicating that the user-supplied data base access function SHIPDATA can be used to produce new members of the BUILDINGS set. This theorem would be used in a derivation, as in the previous example, by creating a subproblem consisting of an instance of the theorem's antecedent. SHIPDATA will be called to produce in the KVISTA a new member of KEYED.APPLICATIONS that will provide the bindings needed to solve the subproblem. A new member of the BUILDINGS set will then be created, as before, to supply the desired bindings for the original query.

In this chapter, we will describe the major constituents of the deduction component and how they interact to answer queries. We begin with a discussion of our concept of "parity", a description of the case analysis tree that defines the alternative answers being constructed, and a presentation of the flow of control in the deduction executive.

B. ELEMENT PARITY

The derivational and retrieval machinery in the deduction component does not require that queries and KVISTA facts and theorems be translated into a canonical form such as prenex normal form or clause form. This flexibility saves statement translation time, reduces the need to keep both an 'internal' and an 'external' form of statements, and allows a user to make entries into the knowledge base in a form that is most advantageous for the class of queries to be answered.

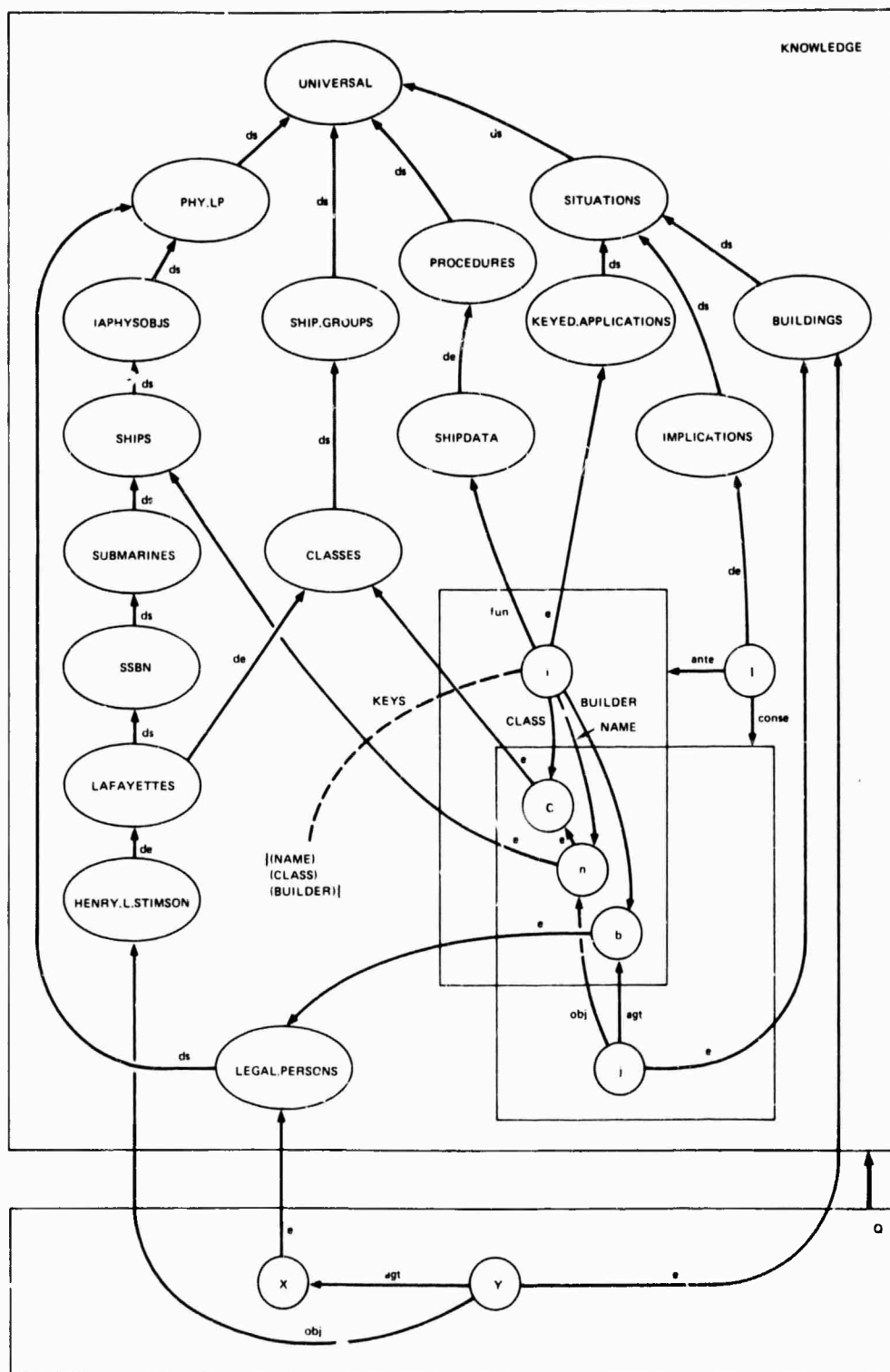


FIGURE XII-4 AN EXAMPLE QUERY, "WHO OWNS THE HENRY.L.STIMSON?", WHOSE ANSWER IS EXTERNALLY DERIVABLE

Therefore, it is necessary to be able to work with arbitrary nestings of negations, implications, and disjunctions containing arbitrary quantifiers both in the KVISTA and in the QVISTA.

One of the problems that this lack of canonicalization presents is determining whether a variable would be universally or existentially quantified if the statement in which the variable appears were transformed so that all the quantifiers occurred at the beginning of the statement (i.e., if the statement were put into prenex normal form). This information can be used, for example, when matching (i.e., unifying) two structures to reject matches that would require binding two distinct existentially quantified elements.

A second problem is that of determining the 'logical sign' (positive or negative) that a KVISTA or QVISTA element has. An element's logical sign corresponds to the sign that the term in which the element occurs would have if the statement in which the element appears were put into disjunctive normal form. This information is important since a QVISTA element and its binding must have the same logical sign. When the derivational machinery is looking for an element in a KVISTA theorem that could produce a binding for some given QVISTA element, it can determine that the derivation cannot possibly produce a binding for the given QVISTA element if the theorem's element does not have the same logical sign.

As an example, elements in the consequent of an implication that is visible in the KVISTA have a positive logical sign and can produce bindings for elements that are visible in the QVISTA (using a derivation that proves an instance of the implication's antecedent), and elements in the antecedent of the implication have a negative logical sign and can provide bindings for elements in a negation space that is visible in the QVISTA (using a derivation that proves the negation of the implication's consequent).^{*} Similarly, a QVISTA element will require a binding that is either visible in the KVISTA or in a negation space that is visible in the KVISTA, depending on how the QVISTA is embedded in negations, implications, and disjunctions.

The 'implicit existential' representation of quantification that we are using (See Chapter V, Section E.2.c) has the interesting property that in the KVISTA all existentially quantified elements have a positive logical sign and all universally quantified elements have a negative logical sign. This correspondence can be understood intuitively by observing that negation changes a universal quantifier into an existential quantifier (and vice versa), and that all universal quantification in our representation is derived from negated existential

^{*} A space being 'in a vista' means that the space is one of the spaces on the list that defines the vista. An element being 'visible in a vista' or 'in the top level of the vista' means that the element is in a space that is in the vista. An element being 'embedded in a vista V' means that the element is not visible in the vista and that the vista of the space that the element is in includes one of the spaces that is in vista V. Hence, for example, the consequent of an implication can be 'visible in the KVISTA', but the elements of the consequent would be 'embedded in the KVISTA'.

quantification. Similarly, with respect to matching QVISTA structures against KVISTA structures, QVISTA elements with a positive logical sign can be thought of as being universally quantified, and QVISTA elements with a negative logical sign can be thought of as being existentially quantified.

Hence, a single device can be used to deal with the two problems discussed above. Namely, functions are available for computing a 'parity' of either 'positive' or 'negative' for each element that is either visible or embedded in either the KVISTA or the QVISTA. Parity corresponds to logical sign and is defined as follows. All elements in the vista have positive parity. All elements in a negation or antecedent space have parity opposite that of the space (supernode) they are in. All elements of any other space (such as a disjunct or consequent space) that are not also elements of a negation space or an antecedent space have parity the same as that of the space they are in.

Parity is used to ensure that anticipated bindings will be legal with respect to quantification and that both elements in a binding will either be visible in the vista or will be in a negation space that is visible in the vista.

C. THE ENVIRONMENT TREE

The deduction component proceeds by growing a case analysis search tree each node of which represents a set of choices, assumptions, and subproblems called an 'environment'. All of the retrieval and deduction activities are done with respect to some environment in this tree.

A typical choice that causes creation of a new environment is the binding of a QVISTA element. Such a choice can be used to derive a contradiction or to restrict possible bindings of other QVISTA elements and therefore is a 'case' that must be considered separately from situations where a different binding is selected for the same QVISTA element. Another typical choice that causes the creation of a new environment is that of a derivational strategy for determining bindings for some construct such as an implication occurring in the QVISTA. The selected strategy may, for example, create a subproblem in which an implication's antecedent is assumed to be true and bindings are sought for the implication's consequent.

Subproblems are formed by adding newly created spaces called "extension spaces" to QVISTA. Similarly, assumptions are made and the results of derivations are stored in newly created extension spaces added to KVISTA. At any given time, the extension space most recently added to KVISTA or QVISTA in a given environment is called the "current" KVISTA or QVISTA extension space for that environment, and the most recently added set of spaces is called the current KVISTA or QVISTA extension vista.

Each time the deduction component is called with a new query, it attaches an empty space to KVISTA and considers it to be the current extension space and the current extension vista to KVISTA. Whenever a new net element is derived in the KVISTA, it is added to the current KVISTA extension space. When a subproblem is being created that involves making assumptions, a new KVISTA extension vista is created and the assumptions are made in the new vista. When the subproblem is completed, the new extension vista is deleted from KVISTA, thereby removing the assumptions and any results derived from them.

The current QVISTA extension vista contains the most recently created subproblems, and bindings are always being sought for the net elements in the current QVISTA extension vista. Associated with each QVISTA extension space is the KVISTA that was current when the extension space was created. Bindings for elements in a QVISTA extension space must be elements of the KVISTA associated with the extension space. This restriction prevents the use during the solution of a subproblem of derived results that depend on assumptions made after the subproblem is created. Initially, all of QVISTA is considered as the current QVISTA extension vista.

Extension spaces are added to KVISTA and QVISTA with respect to an environment, so that KVISTA and QVISTA trees of spaces are grown that map onto the environment tree. The network partitioning facilities provide a natural and efficient mechanism for administering these alternative cases, including their subproblems, assumptions, and derived results.

Extension spaces would be used, for example, to find bindings for an implication occurring in the QVISTA. A typical derivation would assume the implication's antecedent in a newly created KVISTA extension space and put a copy of the consequent into a newly created QVISTA extension space. Bindings would then be sought for the consequent copy, and when a complete set was found, the extension spaces created for this subproblem would be deleted from the KVISTA and QVISTA, a copy would be created of the QVISTA implication in the current KVISTA extension space, and bindings would be formed between the elements of the QVISTA implication and its newly derived KVISTA copy.

An environment is represented by an association list that pairs variable names with their values. A new environment is created as an offspring of an existing environment in the tree. The new environment effectively 'inherits' its own copy of each of the variable values from its parent. Whenever possible, the values are not copied and changes in the new environment are made in such a manner as not to affect the values in the parent. For example, if a value is a list and is not copied, then in the offspring environment elements might be added to the front of the list, but the existing list and its elements would not be changed.

In the version of the deduction component described in this chapter, the following variables are included in an environment:

- (1) BINDINGS -- The list of bindings of QVISTA elements to KVISTA elements.

(2) QVISTA -- The list of spaces in QVISTA. This list includes all the extension spaces that are a part of QVISTA in this environment. The first element of this list is the current QVISTA extension space.

(3) QVISTA.EXTENSION.VISTAS -- A list of the extension vistas that are part of QVISTA in this environment. The list is ordered so that its first element is always the current QVISTA extension vista.

(4) KVISTA -- The list of spaces in KVISTA. This list includes all the extension spaces that are a part of KVISTA in this environment. The first element of this list is the current KVISTA extension space.

(5) KVISTA.EXTENSION.VISTAS -- A list of the extension vistas that are part of KVISTA in this environment. The list is ordered so that its first element is always the current KVISTA extension vista.

(6) EXTRACTED.Q.ELEMENTS -- The list of QVISTA elements that have been extracted in this environment and its ancestors (see Section K on the QVISTA Extractor).

(7) CANDIDATE.GENERATORS -- The generator functions that produce possible bindings for QVISTA elements (See Section D on the Executive). An offspring environment that inherits a generator from its parent environment must effectively have a copy of the generator, since the generator may be independently pulsed in both environments. The 'spaghetti stack' features of INTERLISP (Teitelman, 1975) allow such 'copying' to be done efficiently.

(8) WAITING.Q.ELEMENTS -- The list of QVISTA elements whose candidate generators are waiting for new bindings to occur. No attempt will be made to find a binding for a QVISTA element while it is on this list.

(9) BINDING.DEMONS -- The lists of demons that are associated with each unbound QVISTA element. When a QVISTA element is bound, each of the demons associated with it is given control. In the version of the deduction component described in this chapter, these demons are placed on QVISTA nodes by generators of candidate bindings when those generators cannot proceed until further bindings are made. As is the case with generators of candidate bindings, an offspring environment that inherits demons from its parent must effectively have a copy of the demons so that they can be independently activated in both environments.

D. THE EXECUTIVE FOR THE DEDUCTIVE COMPONENT

The facilities that we have designed for the deductive component allow for a variety of control and selection strategies to guide the search for bindings. We present in this section a simple control strategy that we will assume to be in effect for our discussions in this chapter.

The deduction component is a generator function and it depends heavily on the use of generator functions (See Teitelman, 1975, Section 12). Generator functions are designed to produce sequentially members of some set (as defined by the parameters of each call). When a generator function is called, it creates an entity called a 'generator' that maintains its own data and control environment and can be 'pulsed' (i.e., restarted) an arbitrary number of times. Each time a generator is pulsed, it returns as a value a member of the set it is generating. Generators are a useful control device when some unknown number of a set's members need to be computed.

Control begins in the executive at step SELECT.ENVIRONMENT (shown below) with a single node in the environment tree. In that environment, which is 'active', the bindings list is initialized to ((ANSWER . YES)), the entire QVISTA is considered to be the current QVISTA extension space, and an empty extension space has been created and added to KVISTA.

The deduction component proceeds by selecting an environment and a QVISTA element. It then uses a 'candidate generator' function to produce KVISTA elements that are potential bindings for the selected QVISTA element in the selected environment. A function called RAMIFY is given each potential binding suggested by the candidate generators to determine what other bindings would be directly implied by the suggested binding. For example, a binding for an arc implies bindings for the arc's from-node and to-node. If a binding is implied that contradicts an existing binding, then RAMIFY will reject the potential binding.

Once a potential binding has been accepted by RAMIFY, control is passed to the Binder to make the binding. If either the selected QVISTA element or the potential binding are elements of an implication, disjunction, or negation, then a derivation may be required before the binding can be made. The Binder tests to determine whether a derivation is required. If not, it makes the bindings; if so, it calls the KVISTA Extractor and/or the QVISTA Extractor to create the subproblems and make the assumptions that define the derivation. If an answer to the original query has been produced when control is returned to the top level, then a bindings list is generated; otherwise, the selection cycle is repeated.

"No" answers to questions are produced by creating an offspring environment of the top (first) node in the environment tree. In that environment, the value of ANSWER on the bindings list is changed to NO, and a new QVISTA is created containing only a negation relation. The

elements of this relation's negation space are all the elements of the original QVISTA. This offspring environment can be created and it or any of its offspring can be selected in step SELECT.ENVIRONMENT at any time before a bindings list has been generated with a "Yes" answer pair.

The following is a description of basic control cycle of the deduction component:

SELECT.ENVIRONMENT:

Select an 'active' environment and call it "CURRENT.ENVIRONMENT". If no active environments remain in the tree, then generate a final bindings list consisting of ((ANSWER . UNKNOWN)). If an active environment is found and selected, then continue.

GENERATE.CANDIDATE:

Select an unbound QVISTA element for which a binding is to be sought, and call the selected element "Q.SELECTION". If a generator of candidate bindings for Q.SELECTION does not exist in CURRENT.ENVIRONMENT, then create one. Pulse the candidate generator to produce a candidate binding for Q.SELECTION. If the generator produces a candidate binding, then go to step PROCESS.CANDIDATE.

If either Q.SELECTION is in the top QVISTA level or Q.SELECTION is the only element in a negation space that is in the top QVISTA level, then deactivate CURRENT.ENVIRONMENT (because there is no possible solution in CURRENT.ENVIRONMENT). Go to step SELECT.ENVIRONMENT.

PROCESS.CANDIDATE:

Call the candidate binding produced by the generator "TARGET.ELEMENT", and apply RAMIFY to Q.SELECTION and TARGET.ELEMENT in CURRENT.ENVIRONMENT. If RAMIFY determines that Q.SELECTION cannot be bound to TARGET.ELEMENT, then go to step GENERATE.CANDIDATE.

If RAMIFY does not reject the binding of Q.SELECTION to TARGET.ELEMENT, create an offspring environment of CURRENT.ENVIRONMENT for TARGET.ELEMENT, call the offspring "NEW.ENVIRONMENT", and give control to the Binder. If, when

the Binder returns control, not all the QVISTA elements have been bound in NEW.ENVIRONMENT, then go to step SELECT.ENVIRONMENT.

If all QVISTA elements have been bound in NEW.ENVIRONMENT (i.e., a solution has been found), then generate NEW.ENVIRONMENT's bindings list. If a "Yes" answer was produced, then the deduction component can be pulsed again to produce another set of bindings. If it is pulsed again, then deactivate NEW.ENVIRONMENT and go to step SELECT.ENVIRONMENT.

The basic goal of the QVISTA element selection process that occurs in step GENERATE.CANDIDATE is to select the QVISTA element that will have the minimum number of candidate bindings (i.e., the most constrained element), and thereby minimize the number of cases the executive must consider. This selection is necessarily a guess that can be guided by heuristics such as "Select a node having an outgoing arc whose to-node is bound to a node with a small number of incoming arcs". The selector considers only those elements that are in or are embedded in the current QVISTA extension vista and that are not on the environment's WAITING.Q.ELEMENTS list. (See Section L on Procedural Augmentation for a discussion of the WAITING.Q.ELEMENTS list). A node cannot be selected until the to-node of one of its outgoing arcs is bound, and an arc cannot be selected until either its from-node or its to-node is bound, since those bindings are needed to provide an index into the QVISTA. Nodes that represent disjunctions, negations, or implications, their outgoing arcs, their case arcs, and the to-nodes of their case arcs are not selected because they receive bindings from the Binder as a by-product of binding the elements inside the disjunction, negation, or implication.

E. GENERATING CANDIDATE BINDINGS FOR A SELECTED QVISTA ELEMENT

When the executive selects a QVISTA element to bind, it pulses a generator that produces KVISTA elements having the same parity as the QVISTA element that are potential bindings for the QVISTA element. The standard generator function that is used to produce the potential bindings uses the indexing features of the semantic net in a straightforward way to find potential bindings. A collection of special purpose generator functions also is available that will be used in preference to the standard generator function whenever possible. These special-purpose functions will be discussed below in Section L. In this section we will describe the standard function that is used when no others apply.

The standard candidate generating function works in the following manner. If the selected QVISTA element is a node, say QN0, then each outgoing arc QN0--R-->QN1 (where R is any relation) from node QN0 that has a bound to-node is an index to potential bindings. In particular, if node QN1 is bound to KVISTA node KN1, then the from-node of each incoming arc to node KN1 with relation R is a potential binding for node QN0. If the indexing arc's relation is e or s, then the special 'chaining' functions described below in Section H are used to produce derived incoming e or s arcs to node KN1.

If the selected QVISTA element is an arc QN0--R-->QN1 (where R is any relation) and node QN0 is bound to node KN0, then each outgoing

KVISTA arc from node KNO with relation R is a candidate match for the selected arc. If the selected arc's relation is s or e, then the special 'chaining' functions are used to produce derived outgoing e or s arcs from node KNO. If node QNi is bound, then potential bindings can also be found in the same manner as if the arc's from-node had been selected (i.e., the selected arc is an index).

The order in which candidate bindings are generated is an important factor in the effectiveness of the procedure. In general, candidates that do not require a derivation are preferred. A useful first-order heuristic is to sort the candidates based on their level of embedding in KVISTA. Candidates at the top level require no derivation and are generated first. Each level of embedding implies the need for a derivation to "extract" the candidate. Hence, the level of embedding provides a first-order approximation to the number of derivational steps that will be required.

F. RAMIFICATIONS OF A PROPOSED BINDING

RAMIFY is a function that plays a role similar to that of unification in predicate calculus theorem provers in that it determines the set of 'substitutions' necessary for a match to occur. It takes as input a QVISTA element and a KVISTA element and determines the set of bindings, couplings, and instantiations that would be implied by binding the QVISTA element to the KVISTA element. For example, if a node representing an element of the set of owning situations is bound to a

QVISTA node, then typically that binding will imply bindings for the QVISTA node's e arc to OWNINGS, its agt and obj case arcs, and to-nodes of the case arcs'. If an inconsistency or 'illegal' binding, coupling, or instantiation is implied (for example, two different bindings are implied for the same element), then RAMIFY will indicate that the input KVISTA element is not a possible binding for the input QVISTA element.

A 'binding' is a pairing of a QVISTA element with a KVISTA element of equal parity. An 'instantiation' is a pairing of a KVISTA element having negative parity with a KVISTA element having positive parity (i.e., an instantiation of the universally quantified negative parity KVISTA element). A 'coupling' is any other legal pairing of QVISTA and KVISTA elements (typically a requirement that two universally quantified variables must take the same binding or instantiation).

A binding of some QVISTA element Q_i to some KVISTA element K_i is legal only if it satisfies the following requirements. Let KV denote the KVISTA associated with the QVISTA space that Q_i is an element of. Then, element K_i must either be a top level element of KV , an element of a negation space that is a top level element of KV , or the current KVISTA extension space must be a member of KV .

All other pairings are legal except those that combine a positive parity KVISTA element with a negative parity QVISTA element, a positive parity KVISTA element with another positive parity KVISTA element, or a negative parity QVISTA element with another negative parity QVISTA

element (i.e., if it pairs two distinct existentially quantified elements).

RAMIFY proceeds by applying the following rules to each node N_i that must be paired with a node N_j . For each such pair, first consider each outgoing case arc $N_j \text{--} Rx \text{--} \rightarrow N_k$ from node N_j for any case relation Rx . If node N_i also has an outgoing case arc $N_i \text{--} Rx \text{--} \rightarrow N_l$, then arc $N_i \text{--} Rx \text{--} \rightarrow N_l$ must be pairable with arc $N_j \text{--} Rx \text{--} \rightarrow N_k$ and node N_l must be pairable with node N_k . Second, check to see if the entity represented by node N_i is constrained to be an element (or a subset) of a set that is disjoint from any set that the entity represented by node N_j is constrained to be an element (or a subset) of.

An inconsistency is reported if a universally quantified element (i.e., a positive parity QVISTA element or a negative parity KVISTA element) is forced to have two different bindings or instantiations, or if a node is shown to represent a member (subset) of a set that is disjoint from a set that the entity represented by the node's binding is a member (subset) of.

G. THE BINDER

The Binder described in this section is called when a QVISTA element has been selected in some environment, a candidate binding (i.e., a target element) in the KVISTA has been found, and RAMIFY has been successfully applied to the selected QVISTA element and the candidate binding. The rules described below make the bindings determined by RAMIFY if a derivation is not required. If a derivation is required before the bindings can be made, then the rules invoke the QVISTA Extractor or the KVISTA Extractor to carry out the derivation.

The rules used by the Binder can be thought of in propositional form as being the following:

- "x" is implied by "x".
- "x OR y" is implied by "x".
- "y IMPLIES x" is implied by "x".
- "(x AND y) IMPLIES z" is implied by "~x".
- "~(x AND y)" is implied by "~x".

A detailed description of the rules as actually applied by the Binder follows.

1. QVISTA TOP LEVEL ELEMENTS

Consider the case where the selected QVISTA element is in the top level of the QVISTA. If the target element is a top-level KVISTA element, then in the offspring environment created for the target element make the bindings determined by RAMIFY (see the PROCESS.CANDIDATE step of the basic control cycle, described above in

Section D on the Executive). If the target element is not a top level KVISTA element, then call the KVISTA Extractor to derive a copy of the target element in the top level of the KVISTA that can be a binding for the selected QVISTA element.

2. QVISTA DISJUNCTIONS

Consider the case where the selected QVISTA element is in some disjunct space DS. If bindings have been found for all the elements of space DS, then bindings can be assigned to the entire disjunction (i.e., "x OR y" is implied by "x").

The Binder determines whether all elements in space DS received bindings to top-level KVISTA elements during RAMIFY. If so, then in the offspring environment created for the target element it makes the bindings produced by RAMIFY and assigns dummy bindings to the disjunction and to all elements in the other disjuncts. If not all the elements in space DS received bindings to top-level KVISTA elements during RAMIFY, then the QVISTA Extractor is called to carry out a derivation in which bindings are sought for the elements of space DS while assuming the negation of each of the disjunction's other disjuncts (using the rule that "x OR y" can be proved by assuming " \sim y" and proving "x").

3. QVISTA IMPLICATIONS

a. CONSEQUENT MATCH

Consider the case where the selected QVISTA element is in some consequent space CS. If bindings have been found for all the elements of space CS, then bindings can be assigned to the entire implication (i.e., "y IMPLIES x" is implied by "x").

The Binder determines whether all elements in space CS received bindings to top level KVISTA elements during RAMIFY. If so, then in the offspring environment created for the target element, it makes the bindings produced by RAMIFY and assigns dummy bindings to the implication, the antecedent space, the consequent space, and to all unbound elements in the antecedent. If not, all the elements in space CS received bindings to top level KVISTA elements during RAMIFY, then the QVISTA Extractor is called to carry out a derivation in which bindings are sought for the elements of space CS while assuming the disjunction's antecedent (using the rule that "x IMPLIES y" can be proved by assuming "x" and proving "y").

b. ANTECEDENT MATCH

Consider the case where the selected QVISTA element is in some antecedent space AS. If the bindings found for the elements of space AS include all the elements of a negation space, then bindings can be assigned to the entire implication [i.e., "(x and y) IMPLIES z" is implied by " \sim x"].

The Binder determines whether during RAMIFY all elements in some top-level KVISTA negation space either became the bindings for elements of the antecedent space AS or were instantiated to top level KVISTA elements. If so, then in the offspring environment created for the target element it makes the bindings produced by RAMIFY, binds space AS to the KVISTA negation space, and assigns dummy bindings to the implication, the consequent space, and to all unbound elements in the consequent and the antecedent. If during RAMIFY not all the elements in a top level KVISTA negation space were either the bindings for elements of the antecedent space AS or were instantiated to top level KVISTA elements, then the QVISTA Extractor is called to carry out a derivation in which bindings are sought for a negation in which AS is the negation space while assuming the negation of the implication's consequent (using the rule that "x IMPLIES y" can be proved by assuming " \sim y" and proving " \sim x").

4. QVISTA NEGATIONS

Consider the case where the selected QVISTA element is in some negation space NS. If the bindings found for the elements of space NS include all the elements of a negation space, then bindings can be assigned to the entire negation [i.e., " \sim (x AND y)" is implied by " \sim x"].

The Binder determines whether during RAMIFY all elements in some top-level KVISTA negation space either were designated to be the bindings for elements of space NS or were instantiated to top-level

KVISTA elements. If so, then in the offspring environment created for the target element it makes the bindings from RAMIFY, binds space NS and its outgoing e arc to the KVISTA negation space and its outgoing e arc, and assigns dummy bindings to all unbound elements in space NS.

If during RAMIFY not all the elements in a top-level KVISTA negation space were either designated to be bindings for elements of space NS or were instantiated to top-level KVISTA elements, then one of the extractors is called to carry out a derivation as follows. If space NS contains either more than one node or contains arcs that do not share a common from-node, then space NS is considered to contain a conjunction and the QVISTA extractor is called to carry out a derivation in which bindings are sought for the negation of the conjunct containing the selected QVISTA element while assuming the other conjuncts. Otherwise, the KVISTA Extractor is called to derive a copy of the target element in the top level of the KVISTA that can be a binding for the selected QVISTA element.

H. DERIVING ELEMENT-OF AND SUBSET RELATIONS USING TAXONOMIES

Partitioned semantic networks are particularly well suited for representing taxonomies of sets using s, ds, e, and de arcs. The deduction component has special purpose facilities designed to extract set element and subset information from the taxonomy representation. These facilities make use of the following rules.

Disjoint Sets:

The sets represented by nodes Nx and Ny are disjoint if either:

(1) there exist arcs Nx--ds-->Nz and Ny--ds-->Nz for some Nz; or

(2) there exist nodes Nu and Nv such that the Nx set is a subset of the set represented by Nu, the Ny set is a subset of the set represented by Nv, and the Nu and Nv sets are disjoint.

Subsets:

The set represented by a node Nx is a subset of the set represented by a node Ny if either:

(1) Nx and Ny are the same node;

(2) there exists an arc Nx--s-->Ny;

(3) there exists an arc Nx--ds-->Ny; or

(4) there exists a node Nz such that the Nx set is a subset of the set represented by Nz and the Nz set is a subset of the Ny set.

The set represented by a node Nx is not a subset of the set represented by a node Ny if the Nx and Ny sets are disjoint.

Set Elements:

The entity represented by a node N_x is an element of the set represented by a node N_y if either:

- (1) there exists an arc $N_x \text{--}e\text{--} \rightarrow N_y$;
- (2) there exists an arc $N_x \text{--}de\text{--} \rightarrow N_y$; or
- (3) there exists a node N_z such that the N_x set is an element of the set represented by N_z , and the N_z set is a subset of the N_y set.

The entity represented by a node N_x is not an element of the set represented by a node N_y if there exists a node N_z such that entity N_x is an element of the set represented by N_z , and the N_y and N_z sets are disjoint.

The basic function, called S:CHAIN, that makes use of these rules searches for chains of s or ds arcs in the KVISTA from a given from-node and/or a given to-node. For example, if from-node N_x and to-node N_y are given, then S:CHAIN will look at all of the supersets of N_x , all of the supersets of those supersets, etc., until N_y is encountered or until no new supersets can be found. If N_y is encountered, then S:CHAIN will have found a sequence of nodes N_1, N_2, \dots, N_k such that there exist arcs $N_x \text{--}s\text{--} \rightarrow N_1$ (or $N_x \text{--}ds\text{--} \rightarrow N_1$), $N_1 \text{--}s\text{--} \rightarrow N_2$ (or $N_1 \text{--}ds\text{--} \rightarrow N_2$), ..., $N_k \text{--}s\text{--} \rightarrow N_y$ (or $N_k \text{--}ds\text{--} \rightarrow N_y$), and S:CHAIN can conclude that the set represented by N_x is a subset of the set represented by N_y . If only a from-node is given to S:CHAIN, then it acts as a generator of supersets of the given node. If only a to-node is given, then S:CHAIN acts as a generator of subsets of the given node.

Simple functions have been written that use S:CHAIN to generate the sets that a given node is an element of, to determine whether or not a

given node represents an element of the set represented by a second given node, and to determine whether or not the set represented by a given node is a subset of the set represented by a second given node. These functions provide general service facilities to the major constituents of the deduction component. For example, they are used by RAMIFY to test whether the entity represented by the binding of a given QVISTA node could possibly be an element of the set represented by the binding of the to-node of the given QVISTA node's outgoing e arc, and by the generators of candidate bindings for QVISTA nodes.

I. SIMPLIFICATION OF NEGATIONS

The derivational machinery of the deduction component creates subproblems, makes assumptions, and derives new net structures during the course of a derivation. All of the functions that apply the derivational rules assume that whenever a QVISTA or KVISTA negation space is produced that contains only a disjunction, negation, or implication, the following transformations are performed:

(1) If a negation contains only another negation, then delete both negation relations leaving only the elements of the embedded negation space; i.e., " $\sim(\sim x)$ " becomes " x ".

(2) If a negation contains only a disjunction, then delete the negation and disjunction relations and transform each disjunct into a negation; i.e., " $\sim(x \text{ OR } y)$ " becomes " $\sim x$ " and " $\sim y$ ".

(3) If a negation contains only an implication, then delete the negation and implication relations, delete the antecedent space leaving only its elements, and transform the consequent into a negation relation; i.e., " $\sim(x \text{ IMPLIES } y)$ " becomes " x " and " $\sim y$ ".

J. THE KVISTA EXTRACTOR

Typically, KVISTA elements occurring in negations, disjunctions, or implications can be used to produce bindings for QVISTA elements only if suitable derivations are carried out. The derivational machinery applies the rules given in this section to 'extract' such desirable KVISTA elements from negations, disjunctions, and implications so that they can be asserted or denied in the top level of KVISTA. The rules can be thought of in propositional form as being the following:

To extract "x" from "x OR y", prove " $\sim y$ ".
To extract "x" from "y IMPLIES x", prove "y".
To extract " $\sim x$ " from "x IMPLIES y", prove " $\sim y$ ".
To extract " $\sim x$ " from " $\sim(x \text{ AND } y)$ ", prove "y".

When the Binder is unable to bind a selected QVISTA element to a KVISTA target element and the selected QVISTA element cannot be further extracted, then the KVISTA Extractor described in this section is called to carry out a derivation that is directed toward creating subproblems whose solution will allow the creation of the target element either in a top-level KVISTA space or in a negation space that is a top level KVISTA element. For example, if the target element is "x" in the expression "z IMPLIES (x OR y)", then the KVISTA Extractor will initiate a subproblem in which an attempt is made to prove "z" and " $\sim y$ ". Solution of this subproblem will allow "x" to be asserted.

The derivation is carried out in the offspring environment created for the target element (see Section D above on the Executive). It is begun by creating an empty space and calling it the "conclusion space",

making a copy in the conclusion space of the disjunction, implication, or negation relation in the top level of the KVISTA that the target element is embedded in, and calling the copy of the target element "TARGET.COPY". When creating the copy of the disjunction, implication, or negation, if an element was instantiated or coupled by RAMIFY, then the instantiation or coupling is used in the copy.

The appropriate subproblem is created in new QVISTA extension spaces by repeatedly applying the extraction rules given below to the conclusion space as long as any of them are applicable. The new QVISTA extension spaces containing the subproblem are then designated as the current QVISTA extension vista, control is returned to the executive, and whenever the offspring environment is selected, bindings are sought for the subproblem in the QVISTA extension vista.

When bindings for all the elements of the QVISTA extension vista are found, the derivation is completed by deleting the extension spaces added to QVISTA and KVISTA, adding to the current KVISTA extension space the elements from the deleted KVISTA extension spaces, and adding the elements of the conclusion space to the current KVISTA extension space. When adding elements to KVISTA, if the to-node or the from-node of a conclusion space arc is in the QVISTA, then instead of adding the arc to KVISTA, a copy of the arc is added using the binding of the QVISTA from-node or to-node.

1. KVISTA EXTRACTION RULES

a. KVISTA DISJUNCTIONS

If TARGET.COPY occurs in or is embedded in a disjunction that is a node in the conclusion space, then the next step in the extraction process uses the rule that "x" can be proved by knowing "x OR y" and proving "~y". The extraction proceeds as follows. Delete from the conclusion space the disjunction relation that TARGET.COPY occurs in, add the conclusion space to the KVISTA as a new extension space, call the disjunct space that TARGET.COPY occurs in the new conclusion space, create a space containing negations of each of the other disjuncts, and add the newly created space to QVISTA as a new extension space.

b. KVISTA IMPLICATIONS

1. CONSEQUENT MATCH

If TARGET.COPY occurs in or is embedded in the consequent space CS of an implication IMP that is a node in the conclusion space, then the next step in the extraction process uses the rule that "x" can be proved by knowing "y IMPLIES x" and proving "y". The extraction proceeds as follows. Delete from the conclusion space the implication relation that TARGET.COPY occurs in, add the conclusion space to the KVISTA as a new extension space, create a new conclusion space containing the elements of space CS that are not also elements of

IMP's antecedent space, and add IMP's antecedent space to QVISTA as a new extension space.

ii. ANTECEDENT MATCH

If TARGET.COPY occurs in the antecedent space AS of an implication IMP that is a node of the conclusion space and TARGET.COPY does not also occur in the consequent space CS of IMP, then the next step in the extraction process uses the rule that " $\sim x$ " can be proved by knowing " x IMPLIES y " and proving " $\sim y$ ". The extraction proceeds as follows. Delete from the conclusion space the implication relation that TARGET.COPY occurs in. Add the conclusion space to the KVISTA as a new extension space. Create a new conclusion space containing a negation relation whose negation space has as elements those elements of space AS that are not in the overlap with space AS. Then create a space containing the elements in the overlap and a newly created negation relation whose negation space contains those elements of space CS that are not in the overlap with space AS. Then add the newly created space to QVISTA as a new extension space.

c. KVISTA NEGATIONS

If TARGET.COPY occurs in or is embedded in the negation space NS of a negation relation that is a node in the conclusion space, and space NS either contains more than one node or contains arcs that do not share the same from-node (i.e., space NS contains a conjunction),

then the next step in the extraction process uses the rule that " $\sim x$ " can be proved by knowing " $\sim(x \text{ AND } y)$ " and proving " y ". The extraction proceeds as follows. If TARGET.COPY occurs in or is embedded in a negation, disjunction, or implication that is an element of space NS, then delete all elements of space NS except that negation, disjunction, or implication relation. If TARGET.COPY is a node that is an element of space NS, then delete all elements of space NS except TARGET.COPY and its outgoing arcs. If TARGET.COPY is an arc that is an element of space NS, then delete all elements of space NS except the from-node of TARGET.COPY and all the outgoing arcs of the from-node. In all cases, add the elements deleted from space NS to the QVISTA extension space.

2. AN EXAMPLE

Consider how the KVISTA Extractor would behave in the following example situation. Assume that the KVISTA contains the theorem shown in Figure XII-5 and that node "r" in that theorem is the target element. The following is a predicate calculus representation of the theorem:

$$M(k) \text{ OR } (Au)\{[(u \text{ IN } N) \text{ AND } (Ax)[(x \text{ IN } P) \text{ IMPLIES } (Ey)((y \text{ IN } Q) \text{ AND } R(u,x,y))]] \text{ IMPLIES } (Ez)[(z \text{ IN } S) \text{ AND } T(u,z)]\}.$$

To see the outline of the extraction process, consider the following propositional representation of the theorem:

$$M \text{ OR } \{[N \text{ AND } (P \text{ IMPLIES } (Q \text{ AND } R))] \text{ IMPLIES } (S \text{ AND } T)\}.$$

The extraction proceeds by setting up $\sim M$ as a subproblem to allow the conclusion of the remainder of the theorem. $\sim(S \text{ AND } T)$ is then added to

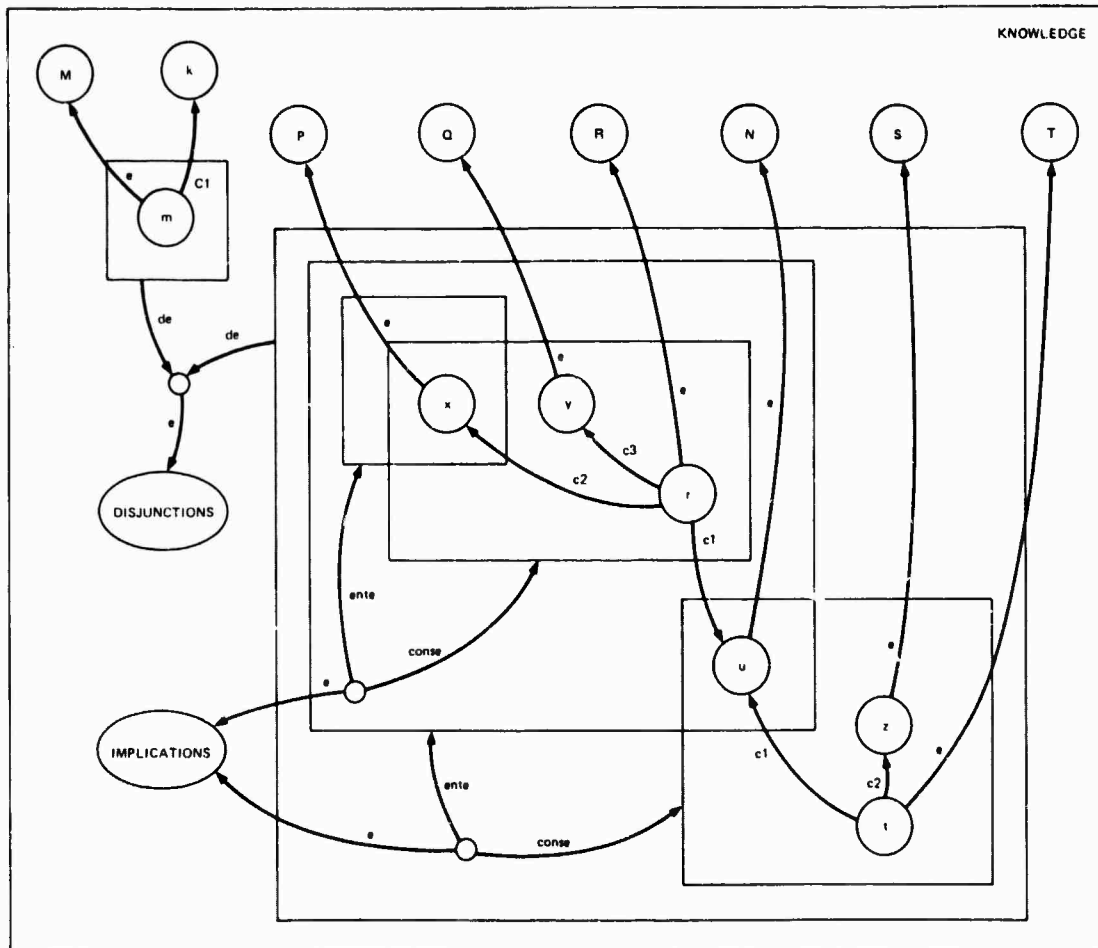


FIGURE XII-5 EXAMPLE KVISTA THEOREM

the subproblem to allow the conclusion $\sim[N \text{ AND } (P \text{ IMPLIES } (Q \text{ AND } R))]$. "N" is then added to the subproblem to allow the conclusion $\sim[P \text{ IMPLIES } (Q \text{ AND } R)]$. This conclusion is transformed into $[P \text{ AND } \sim(Q \text{ AND } R)]$. "P" is then added to KVISTA and "Q" is added to the subproblem to allow the conclusion of $\sim R$. The final conclusion of the derivation is $(\sim R \text{ AND } \sim[N \text{ AND } (P \text{ IMPLIES } (Q \text{ AND } R))])$.

P). Figure XII-6 shows the conclusion spaces and the extension spaces that are created as the fully quantified extraction takes place, and Figure XII-7 shows the QVISTA and KVISTA extension spaces at the end of the extraction. Note that the universally quantified variable "u" becomes a node in a QVISTA extension space and all subexpressions that contain "u" point to that QVISTA node. Placing "u" in the QVISTA allows an arbitrary KVISTA binding to be selected for it and restricts all subexpressions containing "u" to accept that binding.

K. THE QVISTA EXTRACTOR

Finding bindings for negations, disjunctions, or implications occurring in the QVISTA typically requires a derivation. When the Binder is unable to bind a selected QVISTA element to a KVISTA target element because the selected QVISTA element is embedded in a disjunction, implication, or negation, then the QVISTA Extractor described in this section is called to carry out a derivation.

In this derivation, a subproblem is created in which the selected QVISTA element has been 'extracted' and can be bound to the target element. The subproblem is constructed so that its solution will imply bindings for the original disjunction, implication, or negation. For example, if the selected element is "x" in the expression "z IMPLIES (x OR y)", then the extractor will initiate a subproblem in which "z" and "~y" are assumed in the KVISTA and an attempt is made to prove "x" in the QVISTA.

CONCLUSION SPACE

Initially:

$M(k) \text{ OR } (Au)\{[(u \text{ IN } N) \text{ AND } (Ax)[(x \text{ IN } P) \text{ IMPLIES } (Ey)((y \text{ IN } Q) \text{ AND } R(u,x,y))]] \text{ IMPLIES } (Ez)[(z \text{ IN } S) \text{ AND } T(u,z)]\}$

After step 1 (Disjunction Rule):

$(Au)\{[(u \text{ IN } N) \text{ AND } (Ax)[(x \text{ IN } P) \text{ IMPLIES } (Ey)((y \text{ IN } Q) \text{ AND } R(u,x,y))]] \text{ IMPLIES } (Ez)[(z \text{ IN } S) \text{ AND } T(u,z)]\}$

After step 2 (Implication Rule, Antecedent Match):

$\sim\{[(u \text{ IN } N) \text{ AND } (Ax)[(x \text{ IN } P) \text{ IMPLIES } (Ey)((y \text{ IN } Q) \text{ AND } R(u,x,y))]]\}$

After step 3 (Negation Rule):

$\sim(Ax)[(x \text{ IN } P) \text{ IMPLIES } (Ey)((y \text{ IN } Q) \text{ AND } R(u,x,y))]$

After step 4 (negation simplification):

$x, (x \text{ IN } P), \sim(Ey)[(y \text{ IN } Q) \text{ AND } R(u,x,y)]$

After step 5 (Negation Rule):

$\sim R(u,x,y)$

EXTENSION SPACES

KVISTA K1: $x, (x \text{ IN } P)$
(added during step 5)

QVISTA Q1: $\sim M(k)$
(added during step 1; accepts bindings from initial KVISTA)

Q2: $u, \sim[(z \text{ IN } S) \text{ AND } T(u,z)]$
(added during step 2; accepts bindings from initial KVISTA)

Q3: $(u \text{ IN } N)$
(added during step 3; accepts bindings from initial KVISTA)

Q4: $y, (y \text{ IN } Q)$
(added during step 5; accepts bindings from K1 and the initial KVISTA)

Derivation conclusion added to KVISTA after deleting K1:
 $x, (x \text{ IN } P), \sim R(\langle \text{binding of } u \rangle, x, \langle \text{binding of } y \rangle)$

Figure XII-6. EXAMPLE OF KVISTA EXTRACTION

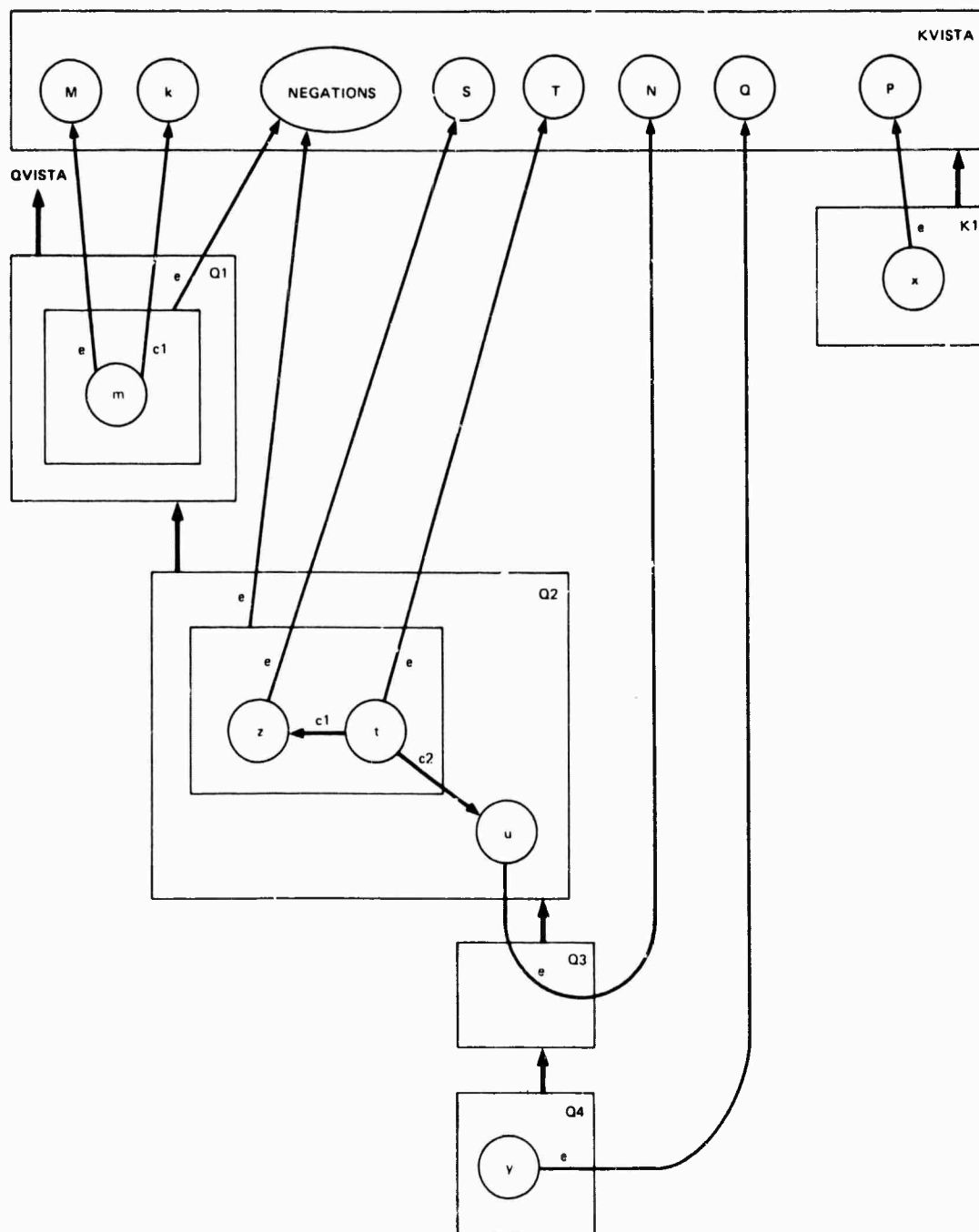


FIGURE XII-7 EXTENSION SPACES FOR KVISTA EXTRACTION EXAMPLE

The rules used by the QVISTA Extractor can be thought of in propositional form as being the following:

To prove "x OR y", assume " \sim y" and prove "x".
To prove "x IMPLIES y", either assume "x" and prove "y" or
assume " \sim y" and prove " \sim x".
To prove " \sim (x AND y)", assume "y" and prove " \sim x".

If the QVISTA element given to the QVISTA Extractor occurs on the environment's list of already extracted QVISTA elements, then the extractor deactivates the offspring environment created for the target element and returns. Otherwise, the extractor adds the QVISTA element to the list and initiates the derivation.

The derivation is carried out in the offspring environment created for the target element. It is begun by calling the top-level disjunction, implication, or negation relation that the selected QVISTA element is embedded in in the "original embedding", making a copy of the original embedding, calling that copy the "current extraction", and calling the copy of the selected QVISTA element "Q.SELECTION.COPY". When making the copy of the original embedding, if an element is bound, then its binding is used in the copy.

The desired subproblems are created and assumptions made by repeatedly applying the extraction rules given below to the current extraction as long as any of them are applicable. The set of new QVISTA extension spaces added to QVISTA by the extraction rules is then designated as the current QVISTA extension vista, control is returned to the executive, and whenever the offspring environment is selected,

bindings are sought for the subproblems in the newly created QVISTA extension vista.

When bindings are found for all the elements of the vista added to QVISTA by the extraction rules, the derivation is completed as follows. The extension spaces added to QVISTA and to KVISTA by the extraction rules are deleted, a copy of the original embedding is added to the current KVISTA extension, and bindings are created between the original embedding and the newly derived KVISTA copy.

1. QVISTA EXTRACTION RULES

a. QVISTA DISJUNCTIONS

If the current extraction is a disjunction, then the next step in the extraction process uses the rule that a disjunction "x OR y" can be proved by assuming " \sim x" and then proving "y". The extraction proceeds as follows. Let DS denote the disjunct space that Q.SELECTION.COPY occurs in. If Q.SELECTION.COPY occurs in or is embedded in a negation, disjunction, or implication that is a node in space DS, then delete from space DS that negation, disjunction, or implication relation, and call the deleted relation the current extraction; otherwise, call Q.SELECTION.COPY the current extraction. Add to KVISTA a new extension space containing negations of each of the disjuncts other than space DS. Then add space DS to QVISTA as a new extension space.

b. QVISTA IMPLICATIONS

i. CONSEQUENT MATCH

If the current extraction is an implication IMP and Q.SELECTION.COPY occurs in or is embedded in the consequent space CS of that implication, then the next step in the extraction process uses the rule that an implication "x IMPLIES y" can be proved by assuming "x" and then proving "y". The extraction proceeds as follows. If Q.SELECTION.COPY occurs in a negation, disjunction, or implication that is a node in space CS, then delete from space CS that negation, disjunction, or implication relation, and call the deleted relation the current extraction; otherwise, call Q.SELECTION.COPY the current extraction. Add to KVISTA as a new extension space the IMP's antecedent space. Then add to QVISTA as a new extension space those elements of space CS that are not in the overlap with the antecedent space.

ii. ANTECEDENT MATCH

If the current extraction is an implication and Q.SELECTION.COPY occurs in or is embedded in the antecedent space AS of that implication and does not also occur in the consequent space CS of that implication, then the next step in the extraction process uses the rule that an implication "x IMPLIES y" can be proved by assuming " \sim y" and then proving " \sim x".

The extraction proceeds as follows. Create a new space QS containing a negation relation whose negation space contains those elements of space AS that are not in the overlap with space CS, and apply the negation simplification rules to the new negation. If after the simplification Q.SELECTION.COPY occurs in a negation, disjunction, or implication that is a node in space QS, then delete from space QS that negation, disjunction, or implication relation, and call the deleted relation the current extraction; otherwise, call Q.SELECTION.COPY the current extraction. Add to KVISTA a new extension space containing those elements that are in the overlap of spaces AS and CS, and a newly created negation relation whose negation space contains those elements of space CS that are not elements of the overlap with space AS. Then add space NS to QVISTA as a new extension space.

c. QVISTA NEGATIONS

If the current extraction is a negation relation and the relation's negation space NS contains not more than one node and no arcs that do not share a common from-node (i.e., space NS does not contain a conjunction), then add the entire current extraction to the current QVISTA extension space since no more extraction need be done. Otherwise, space NS is considered to contain a conjunction and the next step in the extraction process uses the rule that a negated conjunction " $\neg(x \text{ AND } y)$ " can be proved by assuming " x " and then proving " $\neg y$ ".

There are three cases to consider in describing the extraction process. The first case is where Q.SELECTION.COPY occurs in or is embedded in a negation, disjunction, or implication that is a node in space NS. In that case, delete all elements of space NS except that negation, disjunction, or implication relation. Then, create a new space ES containing the current extraction, and apply the negation simplification rules to the current extraction. After the simplification, if Q.SELECTION.COPY occurs in or is embedded in a negation, disjunction, or implication that is a node in space ES, then delete from space ES that negation, disjunction, or implication relation, and call the deleted relation the current extraction; otherwise, call Q.SELECTION.COPY the current extraction. Create a new extension space for KVISTA containing the elements deleted from space NS and then add space ES to QVISTA as a new extension space.

The second case is where Q.SELECTION.COPY is a node that is an element of space NS. In that case, delete all elements of space NS except Q.SELECTION.COPY and its outgoing arcs, and create a new extension KVISTA space containing the elements deleted from space NS. Then add the current extraction to QVISTA as a new extension space and call Q.SELECTION.COPY the new current extraction.

The third case is where Q.SELECTION.COPY is an arc that is an element of space NS. In that case, delete all elements of space NS except the from-node of Q.SELECTION.COPY and all the outgoing arcs of the from-node, and create a new KVISTA extension space containing the

elements deleted from space NS. Then add the current extraction to QVISTA as a new extension space and call Q.SELECTION.COPY the new current extraction.

2. AN EXAMPLE

Consider how the QVISTA Extractor would behave in the following example situation. Assume that the QVISTA contains the theorem shown in Figure XII-5 and that node "r" in that theorem is the selected QVISTA element. We repeat here the predicate calculus representation of the theorem:

$$M(k) \text{ OR } (Au)\{[(u \text{ IN } N) \text{ AND } (Ax)[(x \text{ IN } P) \text{ IMPLIES } (Ey)((y \text{ IN } Q) \text{ AND } R(u,x,y))]] \text{ IMPLIES } (Ez)[(z \text{ IN } S) \text{ AND } T(u,z)]\}.$$

To see the outline of the extraction process, again consider the following propositional representation of the theorem:

$$M \text{ OR } \{[N \text{ AND } (P \text{ IMPLIES } (Q \text{ AND } R))] \text{ IMPLIES } (S \text{ AND } T)\}.$$

The extraction proceeds by making the assumption " $\sim M$ " for use while proving the remainder of the theorem. The assumption " $\sim(S \text{ AND } T)$ " is then made, which can be used while proving " $\sim[N \text{ AND } (P \text{ IMPLIES } (Q \text{ AND } R))]$ ". The assumption "N" is then made to be used while proving " $\sim[P \text{ IMPLIES } (Q \text{ AND } R)]$ ". This subproblem is transformed into " $P \text{ AND } \sim(Q \text{ AND } R)$ ". The assumption "Q" is then made to be used while proving " $\sim R$ ". The extraction ends leaving the subproblems " $\sim R$ " and "P" to be solved. Figure XII-8 shows the current extractions and the extension spaces that are created as the fully quantified extraction takes place, and Figure XII-9 shows the QVISTA and KVISTA extension spaces at the end of the extraction.

CURRENT EXTRACTIONS

Initially:

$M(k) \text{ OR } (Au)\{[(u \text{ IN } N) \text{ AND } (Ax)[(x \text{ IN } P) \text{ IMPLIES } (Ey)((y \text{ IN } Q) \text{ AND } R(u,x,y))]] \text{ IMPLIES } (Fz)[(z \text{ IN } S) \text{ AND } T(u,z)]\}$

After step 1 (Disjunction Rule):

$(Au)\{[(u \text{ IN } N) \text{ AND } (Ax)[(x \text{ IN } P) \text{ IMPLIES } (Ey)((y \text{ IN } Q) \text{ AND } R(u,x,y))]] \text{ IMPLIES } (Ez)[(z \text{ IN } S) \text{ AND } T(u,z)]\}$

After step 2 (Implication Rule, Antecedent Match):

$\sim\{[(u \text{ IN } N) \text{ AND } (Ax)[(x \text{ IN } P) \text{ IMPLIES } (Ey)((y \text{ IN } Q) \text{ AND } R(u,x,y))]]\}$

After step 3 (the portion of the Negation Rule that precedes simplification):

$\sim(Ax)[(x \text{ IN } P) \text{ IMPLIES } (Ey)((y \text{ IN } Q) \text{ AND } R(u,x,y))]$

After step 4 (the remainder of the Negation Rule):

$\sim(Ey)[(y \text{ IN } Q) \text{ AND } R(u,x,y)]$

After step 5 (Negation Rule):

$\sim R(u,x,y)$

EXTENSION SPACES

QVISTA	Q1:	$x, (x \text{ IN } P)$ (added during step 4; accepts bindings from K3, K2, K1, and the initial KVISTA):
	Q2:	$\sim R(u,x,y)$ (added during step 5; accepts bindings from K4, K3, K2, K1, and the initial KVISTA):
KVISTA	K1:	$\sim M(k)$ (added during step 1)
	K2:	$u, \sim(Ez)[(z \text{ IN } S) \text{ AND } T(u,z)]$ (added during step 2)
	K3:	$(u \text{ IN } N)$ (added during step 4)
	K4:	$y, (y \text{ IN } Q)$ (added during step 5)

Figure XII-8. EXAMPLE OF QVISTA EXTRACTION

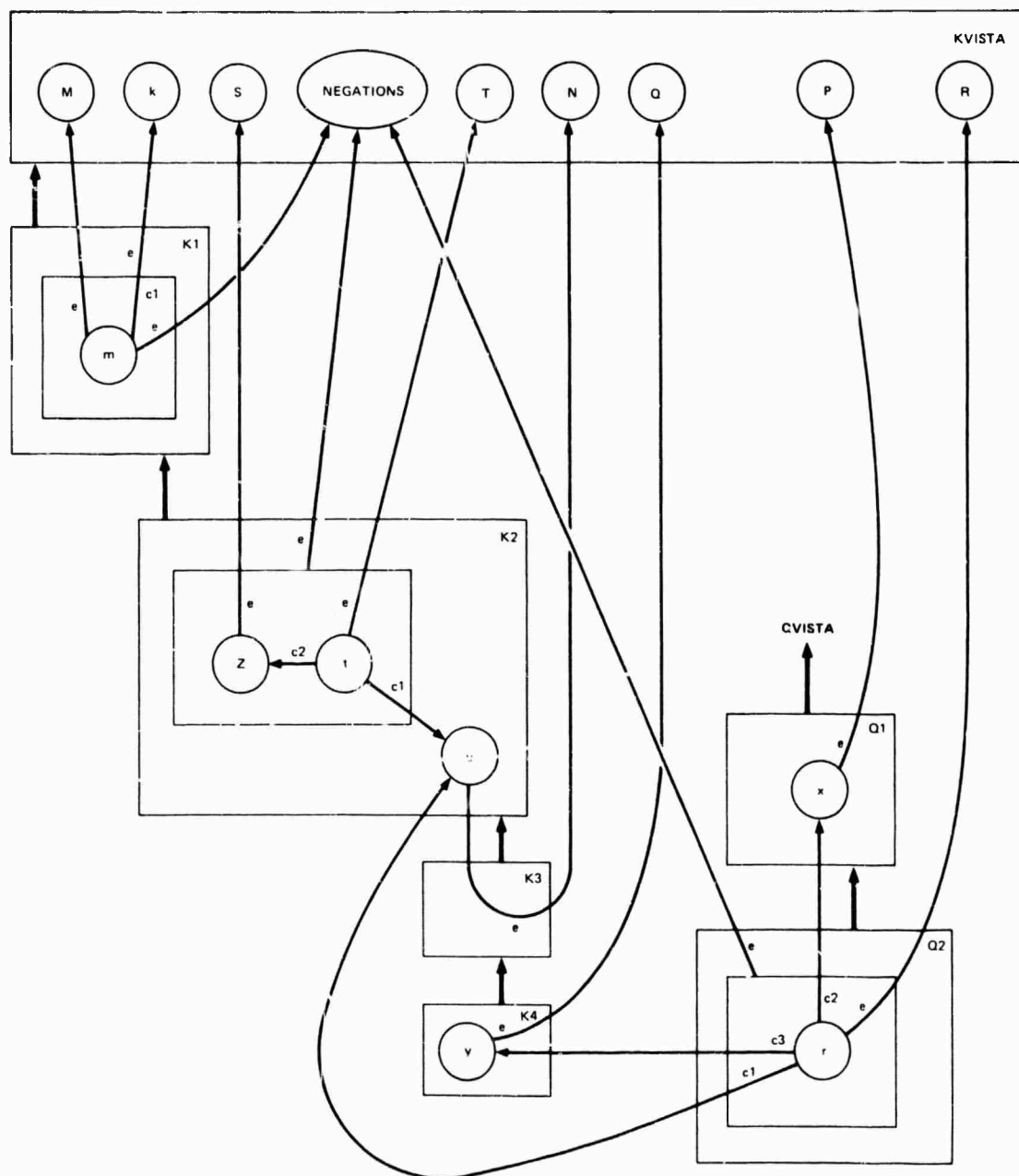


FIGURE XII-9 EXTENSION SPACES FOR QVISTA EXTRACTION EXAMPLE

L. PROCEDURAL AUGMENTATION

We are experimenting with various facilities for allowing the deduction component to be augmented with procedures embodying additional derivational rules and strategies. In our discussion here, we will focus on augmentation in the form of special-purpose functions for generating candidate bindings. These functions are used in preference to the standard function for generating candidates whenever possible. (The standard generator function is described above in Section E). The following subsections describe a basic set of such special-purpose generators.

1. E ARC WITH BOUND NODES

If the selected QVISTA element is an e or s arc having a bound from-node and a bound to-node, then look for a binding by calling one of the derivation functions that chains through the taxonomies. If the derivation function returns a "Yes" answer and the selected arc has positive parity, then create the derived arc in the current KVISTA extension space and generate it as the only candidate binding. If the derivation function returns a "No" answer and the selected arc has negative parity, then create a negation relation in the current KVISTA extension space with a negation space containing only the derived arc and generate the derived arc as the only candidate binding. Otherwise (i.e., if the derivation function returns an "Unknown" answer or the arc's parity prevents a binding), call the standard candidate bindings generator.

2. SETS DEFINED IN THE QVISTA

A query may specify a set and then ask about some property of that set. For example, the query "Did General Electric build any of the nuclear submarines owned by the U.S.?" might be interpreted as "Let US.NUCS be the set of all nuclear submarines owned by the U.S.; did General Electric build any member of US.NUCS?" Such sets that are specified in the query can be arbitrarily created in the KVISTA along with a collection of necessary conditions for membership, a collection of sufficient conditions for membership, or a single condition that is both necessary and sufficient for membership.

A necessary condition for membership in some set represented by a node X is expressed as an implication whose antecedent space contains only a single node with a single outgoing e arc to node X. A sufficient condition for membership in the X set is expressed as an implication whose consequent space contains only a single node with a single outgoing e arc to node X.

The function in the deduction component that creates these KVISTA subsets proceeds as follows. If the selected QVISTA element is a node, QN, that has no outgoing e arcs and that has an outgoing s arc whose to-node is bound to some KVISTA node KX, then create in the current KVISTA extension space a new node, KN, to be the binding for node QN and a new arc KN--s-->KX. In addition, there are three alternative ways in which necessary and/or sufficient conditions can be created for the KN set.

The first option is to create in the current KVISTA extension a copy of any implications in the QVISTA that represent necessary conditions for membership in the set represented by node QN.

The second option can be taken only if KX is the node UNIVERSAL. In that case, copies can be created in the current KVISTA extension space of any implications in the QVISTA that represent sufficient conditions for membership in the QN set.

The third option can be taken only if KX is the node UNIVERSAL. In that case, a copy can be created in the current KVISTA extension space of any double implication in the QVISTA that is both a necessary and a sufficient condition for membership in the QN set.

When making these copies, if an element is bound, then its binding is used in the copy. The generator completes its work by generating a set of bindings for node QN, the outgoing s arc for QN, and all the unbound QVISTA elements that are part of copied implications.

3. APPLICATIONS AND KEYED-APPLICATIONS

Special-purpose generator functions exist to create elements of the APPLICATIONS and KEYED-APPLICATIONS sets when they are needed by calling the appropriate functions. For example, if the KVISTA Extractor adds to the QVISTA the antecedent of the implication shown in Figure XII-10 and the to-nodes of the "addend1" and "addend2" arcs are bound, then the "plus" function can be called to obtain a sum and create in the

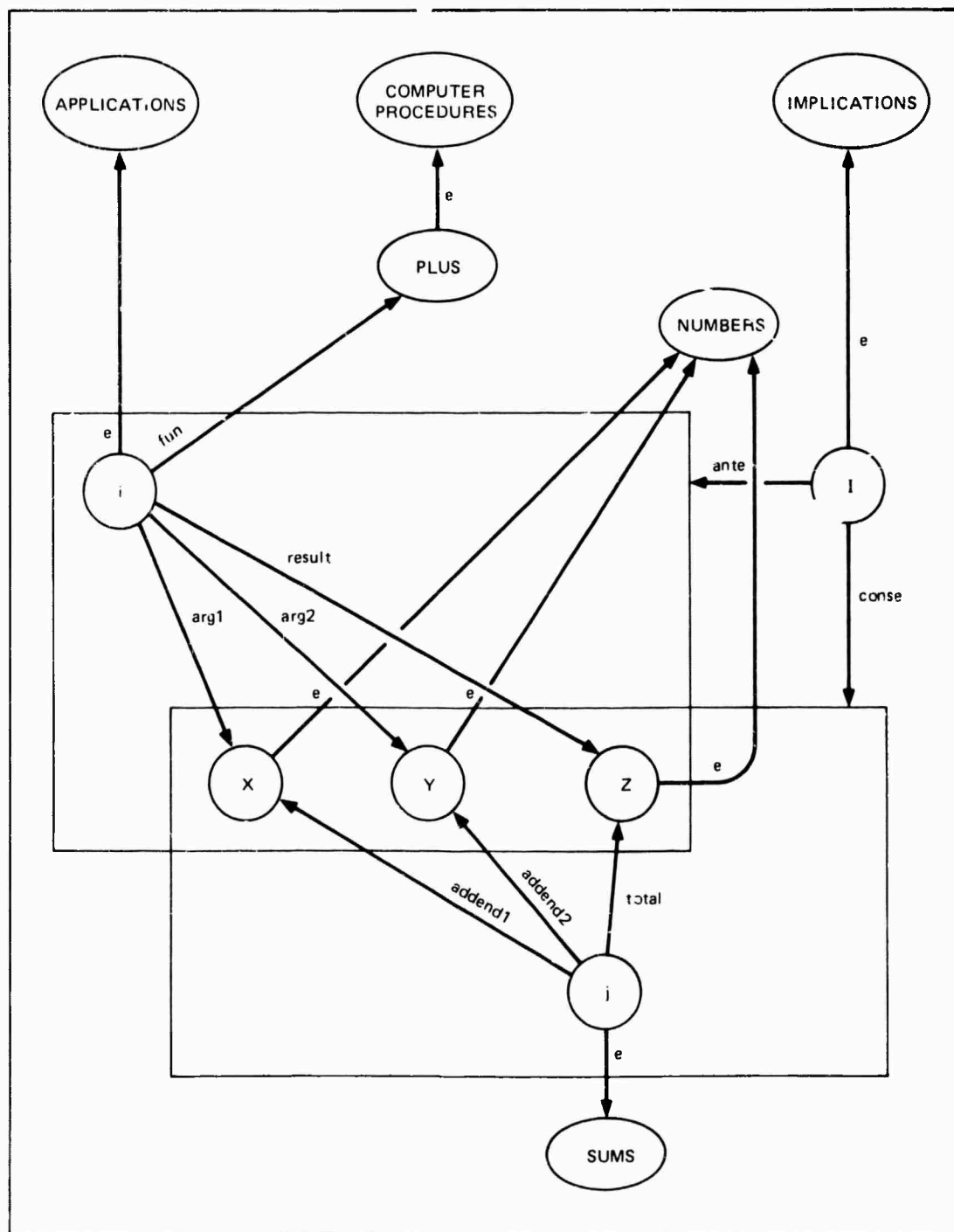


FIGURE XII-10 RELATING SUMS SITUATIONS TO FUNCTION PLUS

KVISTA an element of APPLICATIONS that will provide bindings for the corresponding nodes and arcs in the antecedent. This example is discussed in Chapter V, Section F.2, in connection with the same figure.

When a QVISTA node is selected that has an outgoing e arc whose to-node is bound to APPLICATIONS, a special purpose generator of candidate bindings for APPLICATIONS is called. This generator checks to see if all the to-nodes of the selected node's "ARGi" case arcs are bound. If they are, then it calls the indicated function with the bindings of the "ARGi" arcs' to-nodes as arguments, creates a new element of the APPLICATIONS set in the current KVISTA extension space using the result produced by the function, and generates a binding of the selected QVISTA node to the newly created APPLICATIONS set element. If not all of the to-nodes of the ARGi arcs are bound, then the selected QVISTA node is added to the current environment's WAITING.Q.ELEMENTS and a "demon" is attached to one of the unbound to-nodes.

Demons are INTERLISP forms or stack pointers (Teitelman, 1975, Section 12) that can be associated with any QVISTA element in an environment. Whenever a QVISTA element is bound, any demons that are associated with that element in the current environment are called. The demons attached to the to-nodes of ARGi arcs by the candidate bindings generator of the APPLICATIONS set check to see if the to-nodes of the other ARGi arcs are bound. If they are, the demon removes the from-node of the ARGi arcs from the WAITING.Q.ELEMENTS list. Otherwise, it attaches itself to an unbound to-node of an ARGi arc and pauses.

When a QVISTA node is selected that has an outgoing e arc whose to-node is bound to KEYED-APPLICATIONS, a special-purpose generator of candidate bindings for KEYED-APPLICATIONS is called. The generator conducts a 'key set bindings test' to determine if the to-nodes of all the ARG1 arcs in any of the key sets are bound. If they are, then it calls the KEYED-APPLICATION element's function with an a-list containing the available bindings of ARG1 arcs' to-nodes and indicating with question marks the desired missing bindings, creates a new element of the KEYED-APPLICATIONS set in the current KQVISTA extension space using the results generated by the function, and generates a binding of the selected QVISTA node to the newly created KEYED-APPLICATIONS set element. If not all of the to-nodes of the ARG1 arcs in any key set are bound, then the selected QVISTA node is added to the current environment's WAITING.Q.ELEMENTS and a demon is attached to the to-nodes of at least one ARG1 arc in each key set. These demons when called, repeat the key set bindings test on ARG1 to-nodes. If the to-nodes of all the ARG1 arcs in some key set are bound, then the ARG1 arcs' from-node is removed from the WAITING.Q.ELEMENTS list. Otherwise, demons are attached as before.*

If the function called by the APPLICATIONS or KEY-APPLICATIONS candidate binding generator is a generator, then each time the candidate binding generator is pulsed, it pulses the function to produce a new element of either the APPLICATIONS or KEY-APPLICATIONS set.

 * This KEYED-APPLICATIONS mechanism was motivated by a desire to interface the deduction component with a data base management system. Jonathan Slocum designed and implemented the interface.

4. EFFICIENCY CONSIDERATIONS

The special-purpose generators for the APPLICATIONS and KEY-APPLICATIONS sets are an adequate facility for dealing with 'evaluable predicates' (Green, 1969) in the deduction component. The mechanism is simple, and the representation explicitly indicates the functions, their arguments, their results, and the additional knowledge provided by their results.

However, a significant gain in efficiency can be obtained at the expense of the representation's explicitness by allowing functions for generating candidate bindings to be attached to any KVISTA node that represents a set. These functions would be used as generators of candidate bindings for QVISTA nodes that are constrained to be elements of the set the function is attached to. For example, the APPLICATIONS and KEYED-APPLICATIONS generators would be attached to the APPLICATIONS and KEYED-APPLICATIONS nodes and would be called by this mechanism. This facility would allow computations such as data base accesses, sums, and products to be done directly by the generators without the need to apply the deduction component's derivational machinery to KVISTA theorems. The degree to which one should sacrifice the explicitness and "purity" of the representation with this mechanism in order to gain derivational efficiency, it seems, must be decided by considering the goals for each particular use of the system.

M. TWO EXAMPLES

Consider again how the deduction component would answer the example query "Who built the Henry L. Stimson?" in Figure XII-2. An initial environment, E0, would be created in the environment tree and selected as the "CURRENT.ENVIRONMENT". A QVISTA element, say node Y, would be selected as the "Q.SELECTION", and the standard function for generating candidate bindings would be used to create a candidate bindings generator for Y. The generator would consider as candidate bindings the from-nodes of either incoming obj arcs to node Henry.L.Stimson or incoming "e" arcs to node BUILDINGS. For the KVISTA shown, node B would be generated as the "TARGET.ELEMENT".

RAMIFY would be given the (Y,B) binding and would determine that unique bindings are implied for the remainder of the elements in QVISTA as follows. Since both Y and B have outgoing agt and obj case arcs, Y's case arcs must be bound to B's corresponding case arcs, which implies bindings for the case arcs' to-nodes. In particular, node X must be bound to node General.Dynamics. Since both nodes X and Y have outgoing "e" arcs with bound to-nodes, RAMIFY will test whether the set membership of those nodes is consistent with the set membership of their bindings. This test employs the derivation functions that follow subset chains, and therefore the required "e" arc between General.Dynamics and LEGAL.PERSONS is derived.

RAMIFY would output its set of implied bindings, and control would be given to the Binder. Since both Y and B are in the top level of their vistas, all the bindings would be made and control would return to the executive. When no further unbound QVISTA elements can be found by the executive, the set of bindings with a "Yes" answer would be generated.

Now consider how the deduction component would answer the same example query given the KVISTA shown in Figure XII-3. In this case the candidate bindings generator for node Y would generate node "b" as a candidate. Given the (Y,b) binding, RAMIFY would determine bindings for node Y's outgoing arcs, a binding of node X to node General.Dynamics, and an instantiation of node Z to node Henry.L.Stimson.

Since the candidate binding for node Y (namely, node b) is not in the top level of the KVISTA, the Binder would call the KVISTA Extractor to derive a binding for node Y. Node "b" is the target and it occurs in the consequent of an implication. Hence, the KVISTA Extractor would add an extension vista to QVISTA consisting of one space containing an "e" arc from node Henry.L.Stimson (the instantiation of node Z) to node LAFAYETTES, and would then return control to the executive.

When the executive selects environment E1 as the current environment, the arc Henry.L.Stimson--e-->LAFAYETTES would be selected and the special-purpose candidate bindings generator for "e" arcs with bound to- and from-nodes would be used to find a matching arc. RAMIFY

would not find any additional bindings implied by the binding of the "e" arc, and the Binder would be able to make the arc binding.

The binding of the "e" arc would complete the binding of the extension vista (i.e., the subproblem) added to QVISTA by the KVISTA Extractor, and therefore would cause the QVISTA extension vista to be deleted and an addition to be made to the current KVISTA extension space. The addition would be a copy of the elements in the implication's consequent space with the to-node of the obj arc in the copy being node Henry.L.Stimson. The KVISTA would then contain an explicit match for the QVISTA elements and the deduction component would find the matches as in the first example, binding the QVISTA elements to the newly derived copy of the implication's consequent.

XIII GENERATING VERBAL RESPONSES

Prepared by Jonathan Slocum

CONTENTS:

- A. Introduction
- B. Generation Templates
- C. Noun Phrases
- D. Discussion
- E. Looking Ahead

A. INTRODUCTION

When an input utterance has been analyzed and the semantic content of an appropriate response has been developed (for example, the answer to a question), the problem of formulating this response for presentation to the user remains. It is the responsibility of the generator, interpreting a grammar, to produce an English output from the given semantic information. This program determines exactly how the response may be formulated -- as a noun phrase, a sentence, or a sequence of sentences. It chooses words and phrases with which to express the semantic content, as well as a syntactic frame for their organization, and it produces the response in 'text' form. This text string can be transformed into a sequence of phonemes via a word pronunciation dictionary and output by a VOTRAX speech synthesizer. However, in the current system, no sentence intonation or stress

contouring is performed; that is, the word pronunciation is context-free. Therefore, the production of speech per se is relatively uninteresting and will not be mentioned further.

B. GENERATION TEMPLATES

In the semantic component of the speech understanding system, situations and events are verb-dominated. By this, we mean that events and situations are expressed by means of verbs or verb-like constructs; they take as 'arguments' entities that are usually expressed by nouns or noun-like constructs. As a result, the grammar rules that generate sentences depend primarily on the verb, and secondarily on its arguments.

Even a cursory study of a few hundred English verbs shows that they impose definite, regular constraints on the syntactic forms of their arguments in sentences. These syntactic constraints depend on particular senses of particular verbs; thus, it seems inappropriate to maintain a global, monolithic grammar for the purposes of generation. This fact has not been noted in previous work on language generation in which verbs were studied whose arguments were exclusively noun phrases (NP) or prepositional phrases (PP). However, there are many instances of constructs other than NP or PP. Examples include: choose "to go", authorize "there to be a demonstration", consider "it easy", see "her drive away", imagine "him laughing", find "her murdered", wonder "how to do it", ask "why I believe", suggest "sending them away". In

particular, consider the difference between "stop to help" and "stop helping": the different senses of "stop" demand different syntactic realizations. For this reason, we associate verbs and grammar rules (in the form of templates) with specific word senses (prototypical nodes) in the net.

Our examples will employ simplified semantic net structures. In the net fragment in Figure XIII-1, the U.S. and the U.K. are elements (e) of the set of countries. As agents (agt) they each participate in OWNING situations involving as objects (obj) particular ships; each ship is an element of a particular set [class] of ships; each class is a subset (s) of a particular set [type] of ships; each type is a subset of the set of all ships.

Now, consider the node in Figure XIII-1 labeled S.OWN. This node is the prototypical OWN, in that it incorporates the meaning of the situation of owning (including any semantic constraints on its arguments), and in that all instances of owning situations are related to it. With this node we associate the appropriate verbs (OWN, POSSESS, HAVE, BELONG) and their 'generation templates.' One template will not suffice for all four verbs; for instance, the syntactic subject of the verb BELONG is the OBJECT case argument, while in the other (active) verbs the subject is the AGENT:

AGT owns OBJ ; OBJ is owned by AGT
AGT possesses OBJ ; OBJ is possessed by AGT
AGT has OBJ ; OBJ belongs to AGT

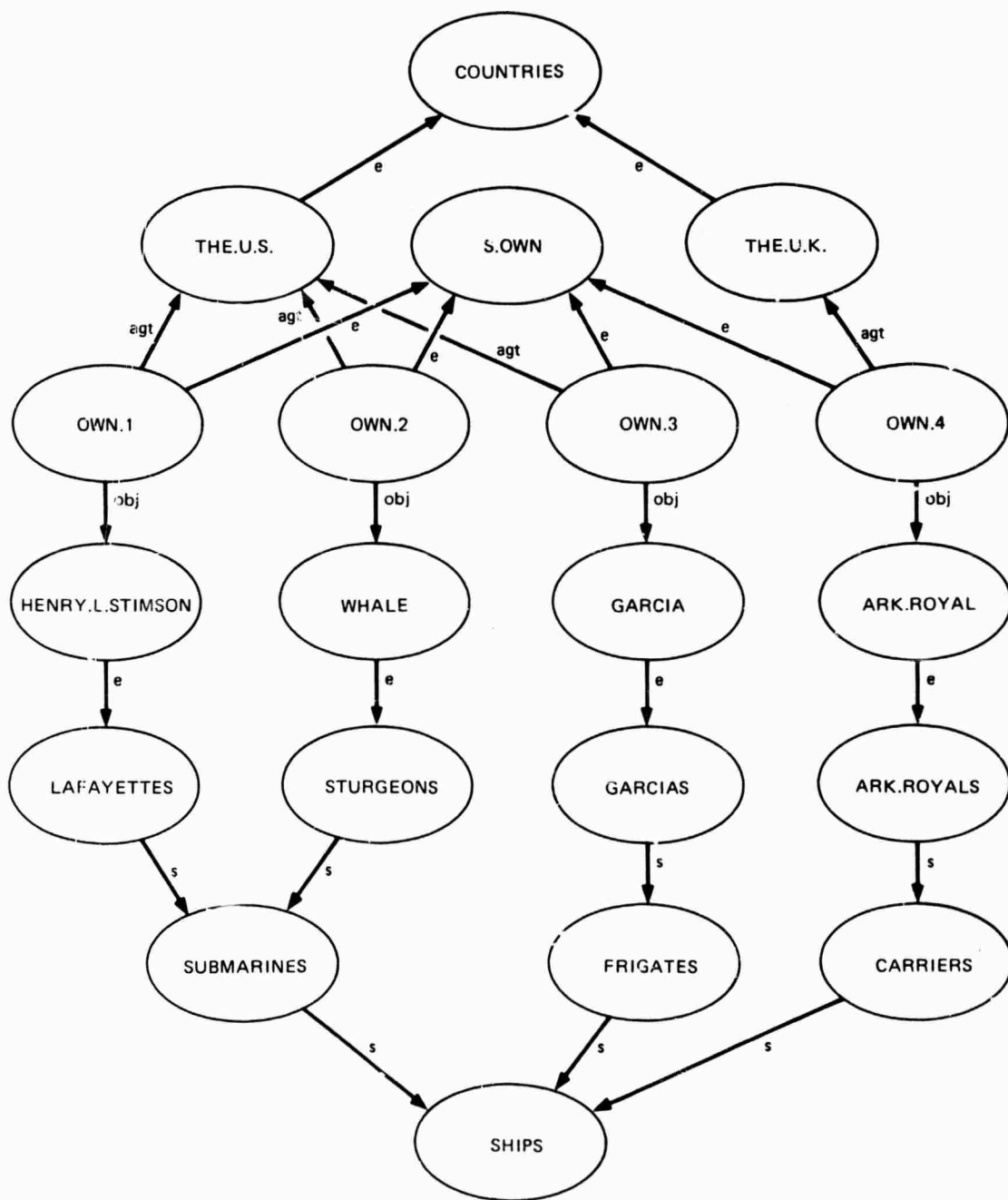


FIGURE XIII-1 FRAGMENT OF A SEMANTIC NETWORK

So we employ the corresponding templates:

```
[OWN ((NP AGT) Vact (NP OBJ)) ((NP OBJ) Vpas BY (NP AGT))]  
[POSSESS ((NP AGT) Vact (NP OBJ)) ((NP OBJ) Vpas BY (NP AGT))]  
[HAVE ((NP AGT) Vact (NP OBJ))]  
[BELONG ((NP OBJ) Vact TO (NP AGT))]
```

A set of templates like these is associated with every 'prototype verb' node in the semantic net. The sentence generation algorithms are then fairly simple (see Figure XIII-2), and constituent functions (e.g., NP) are responsible for controlling subcomponents of the grammar -- generally, through appropriate recursive calls to the interpreter. For example, in order to speak about a particular owning situation (such as OWN.2), we pursue the hierarchy to find the 'canonical' S.OWN, choose a verb (say, BELONG) and an associated template [(NP OBJ) Vact TO (NP AGT)], and generate the constituents:

```
verb [OWN.2 --> S.OWN] --> belong  
template --> [(NP OBJ) Vact TO (NP AGT)]  
(NP OBJ) --> [NP WHALE] --> the Whale  
    Vact --> belongs  
    TO --> to  
(NP AGT) --> [NP U.S.] --> the United States
```

TO GENERATE A RESPONSE:

1. Generate-a-sentence; if succeed, return sentence
2. Generate-an-NP; if succeed, return noun phrase
3. Else FAIL

TO GENERATE-A-SENTENCE:

1. Generate-a-clause

TO GENERATE-A-CLAUSE:

1. Choose-a-verb; if none available, FAIL
2. Choose-a-template; if none available, go to step 1
3. Apply-the-template; if fail, go to step 2
4. Generate-verb-string
5. Concatenate the subject, verb string, and predicate
6. Return results

TO CHOOSE-A-VERB (-NOUN):

1. Retrieve a verb (noun) associated with current node;
if found, return it
2. Take 1 step up hierarchy for new current node
3. If succeed, go to step 1; else FAIL

TO CHOOSE-A-TEMPLATE:

1. Retrieve a template associated with current node and word
2. If found, return it; else FAIL

TO APPLY-THE-TEMPLATE:

1. Initialize RESULTS to be empty
2. If template is empty, return RESULTS
3. Evaluate-first-constituent; if unsuccessful, FAIL
4. Concatenate RESULTS and answer from step 3
5. Discard first constituent from template
6. Go to step 2

TO EVALUATE-FIRST-CONSTITUENT:

1. If constituent is atomic, return it (unevaluated)
2. (Constituent is a list); apply function named as first
item in constituent to network nodes indicated by rest
of constituent
3. Return results of step 2

TO GENERATE-AN-NP:

1. Choose-a-noun; if none available, FAIL
2. Choose-a-template; if none available, go to step 1
3. Apply-the-template; if fail, go to step 2
4. Return results of step 3

Figure XIII-2. THE BASIC GENERATION ALGORITHMS

C. NOUN PHRASES

In the current generator, noun templates are used to control noun phrase generation. Much like verb templates, noun templates order the constituents in the phrase and indicate how each constituent is to be generated by naming a function to be called with the network constituent. For example, with the node WHALE we associate the template [WHALE (THE (N))], which enables us to speak of the particular submarine named "Whale" as "the Whale." Associating, e.g., [STURGEON ((DET) (N))] with the node STURGEONS (the set of all "Sturgeon class" submarines) allows us to speak of an indeterminate member of that class as "a Sturgeon," or of a subset of that class as "the Sturgeons." And associating, e.g., [SUBMARINE ((DET) (N))] with the node SUBMARINES (the set of all submarines) allows us to speak of an indeterminate submarine as "a submarine," or of a subset of submarines as "the submarines." NP templates like this are distributed throughout the network hierarchy. We do not consider this method to be entirely sufficient, but it handles all current requirements.

D. DISCUSSION

In theory, the set of possible English sentences is infinite. The obvious question then arises, "If one tries to account for them with templates, won't there be an infinite number of templates?" The simple answer is, "No, for some of the same reasons that allow a finite grammar to generate an infinite number of strings." Sentences of arbitrary length are produced by arbitrary embedding and arbitrary conjunction, not by including arbitrary numbers of distinct case arguments. Even so, the number of basic syntactic patterns (devoid of case names and particular prepositions) might seem to be extremely large. Evidence, however, is to the contrary. Hornby and his colleagues (1948, 1954) show the number of patterns to be small. The eventual number of templates would appear to be several times the number of patterns, owing to the substitution of particular prepositions for 'prep' in the syntactic patterns, and the assignment of different case names to a particular constituent, depending on the particular verb used.

One may question whether templates should be stored for passives; certainly, they could be derived. On the other hand, neglecting to store them would force us to indicate with each verb (sense), whether it can (or, sometimes, must) be passivized. Specifying 'transitive' is not enough since there are transitive verbs (i.e., verbs that take an object) that cannot be passivized. Since we have to store the information anyway, we can save some code and computing time by storing the passive template.

There are several reasons for generating the verb after the major arguments. First, the subject must be generated so that the verb can be made to agree in number. Second, certain word senses are true of verb-particle combinations while not of the isolated verb. Since particles must appear after objects that are short (like pronouns) but before objects that are long (like noun phrases), the particle must be positioned after the object phrase is generated. Finally, insertion of some adverbials (e.g., "not") requires an auxiliary verb; thus verb generation must follow adverbial generation, so that any use of mid-position adverbials will affect the generation of the verb string.

E. LOOKING AHEAD

There are some sources of potential template proliferation, an important one being the combinatorial arrangements of the case arguments of time, manner, and other adverbials, as well as other (possibly non-adverbial) case arguments like source, goal, and instrument. Some of these arguments are rather constrained in their positions in the sentence, but others may appear almost anywhere:

"Yesterday the ship sailed from the lighthouse to the dock."

"The ship sailed from the lighthouse to the dock yesterday."

"From the lighthouse the ship sailed yesterday to the dock."

It is of course unreasonable to try to cover all these cases with templates; instead we will leave insertion of these adverbial arguments to a single heuristic routine. There are several justifications for this solution, among them: the particular form of the verb cannot be

generated until the subject, object(s), and complement(s) have been generated; these adverbials are so universal as to appear in almost any of the templates and in several possible places; and there are heuristic constraints involved in the placement of arguments.

There are no well-formulated rules accounting for noun phrases in English; indeed, there are few well-established guidelines other than that the hearer must be able to resolve the pronouns and noun phrases to their referents. The speaker should employ anaphora in order to avoid repetition, but only when his 'model of the hearer' indicates that there will be no ambiguity. Problems include: whether to use a proper noun (name) if the referent has one, or whether to employ a pronoun or common noun; in the latter case, which of the available common nouns, what determiners to use, what adjectives and postmodifiers, etc. The generator should piece these constituents together in some reasonable order, performing appropriate lexical transformations, and recursively expanding any constituent phrases.

Some low-power pronominalization rules can be directly incorporated in a grammar -- reflexivization, for example. Otherwise, a grammar should not determine the components of a constituent phrase independent of the conversational context. This situation has not been universally recognized, but it is becoming increasingly clear that a discourse module, operating on some model of the hearer, must be consulted during the generation phase. The generator should pass any 'noun' constituent to the discourse module (perhaps with its recommendation about how to

produce the constituent); the module must determine if a pronoun or bare noun is ambiguous to the hearer, and, if so, what to add to the noun in order to make the desired referent clear. In the future, more general templates, for example,

[(DET) (Adj QUAL) (Adj SIZE) (Adj SHAPE) (Adj COLOR) (N)] ,
may be employed, and a discourse module will decide for each template constituent whether it is to appear in the phrase.

It would appear that, for generation purposes at least, our modular grammar has an important advantage over a 'monolithic' grammar: it clearly indicates the syntactic idiosyncracies imposed by particular word choices. The storage requirements of the two formalisms are probably similar. The modular grammar will probably require more rules, but a monolithic grammar must in turn incorporate many 'applicability tests' for each of its rules. In effect, these tests are precomputed during the construction of the modular grammar. Further research should enable us to verify these claims.

XIV REFERENCES

- Aho, A. V., and Ullman, J. The Theory of Parsing, Translation, and Compiling. Volume 1. Prentice-Hall, Englewood Cliffs, New Jersey, 1972.
- Baker, James K. The DRAGON System -- An Overview. IEEE Transactions on Acoustics, Speech, and Signal Processing, 1975, ASSP-23, 24-29.
- Barnett, Jeffrey A. INFIX LISP for SDC IBM 370 Users. TM-4310, System Development Corporation, Santa Monica, California, May 1973.
- Barnett, Jeffrey A. Speech Understanding Sytem Overview. TM-5732, System Development Corporation, Santa Monica, California, August 1976.
- Barnett, Jeffrey A., and Pinter, Douglas. CRISP: A Programming Language and System. TM-5455, System Development Corporation, Santa Monica, California, December 1974.
- Bates, Madeleine. Syntactic Analysis in a Speech Understanding System. BBN Report 3116, Bolt Beranek and Newman, Cambridge, Massachusetts, August 1975.
- Becker, Richard, and Poza, Fausto. Acoustic Processing in the SRI Speech Understanding System. IEEE Transactions on Acoustics, Speech and Signal Processing, 1975, ASSP-23, 416-426.
- Bernstein, Morton I. Interactive Systems Research: Final Report. TM-5243/004, System Development Corporation, Santa Monica, California, 1975.
- Bobrow, Daniel G., and Fraser, J. Bruce. An Augmented State Transition Network Analysis Procedure. Proceedings of the International Joint Conference on Artificial Intelligence, Washington, D.C., 7-9 May 1969. Edited by Donald E. Walker and Lewis M. Norton. The MITRE Corporation, Bedford, Massachusetts, 1969, 557-568.
- Bobrow, Daniel G., and Wegbreit, Ben. A Model and Stack Implementation of Multiple Environments. Communications of the ACM, 1973, 16, 591-603.

- Bobrow, Daniel G., and Winograd, Terry. An Overview of KRL, a Knowledge Representation Language. Computer Science Laboratory, Xerox Palo Alto Research Center, Palo Alto, California, and Artificial Intelligence Laboratory, Stanford University, Stanford, California, July 1976.
- Bruce, Bertram C. A Model of Temporal References and Its Application in a Question Answering Program. Artificial Intelligence, 1973, 3, 1-26.
- Celce-Murcia, Marianne. Verb Paradigms for Sentence Recognition. American Journal of Computational Linguistics, Microfiche 38, 1976.
- Chafe, Wallace L. Discourse Structure and Human Knowledge. In: Language Comprehension and the Acquisition of Knowledge. Edited by Roy O. Freedle and John B. Carroll. Winston, Washington, D. C., 1972. Pp. 41-69.
- Chafe, Wallace L. Language and Consciousness. Language, 1974, 50, 111-133.
- Chapanis, Alphonse. The Communication of Factual Information Through Various Channels. Information Storage and Retrieval, 1973, 9, 215-231.
- Chapanis, Alphonse. Interactive Human Communication. Scientific American, March 1975, 36-42.
- Charniak, Eugene. Toward a Model of Children's Story Comprehension. AI TR-266, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1972.
- Cheatham, Thomas E., and Sattley, Kirk. Syntax Directed Compiling. AFIPS Conference Proceedings: Spring Joint Computer Conference, 1964, 25, 31-57.
- Cohen, Philip R., and Perrault, C. Raymond. Preliminaries for a Model of Conversation. In Proceedings of the First CSCSI/SCEIO National Conference, Vancouver, British Columbia, Canada, 25-27 August 1976. Pp. 102-111.
- Colby, Kenneth M., Faught, William, and Parkinson, Roger. Pattern Matching Rules for the Recognition of Natural Language Dialogue Expressions. AI-MEMO 234, Artificial Intelligence Laboratory, Stanford University, Stanford, California, June, 1974.
- Cox, D. R. Planning of Experiments. John Wiley, New York, 1958.
- Deutsch, Barbara G. The Structure of Task-Oriented Dialogs. Contributed Papers, IEEE Symposium on Speech Recognition. Carnegie-

- Mellon University, Pittsburgh, Pennsylvania, 15-19 April 1974.
 Edited by Lee D. Erman. IEEE, New York, 1974, 250-254. (a)
- Deutsch, Barbara G. Typescripts of Task Oriented Dialogs. SUR Note 146, Stanford Research Institute, Menlo Park, August 20, 1974. (b)
- Deutsch, Barbara G. Establishing Context in Task-Oriented Dialogs. American Journal of Computational Linguistics, 1975, 4, Microfiche 35.
- Erman, Lee D. An Environment and System for Machine Understanding of Connected Speech. Ph.D. Thesis, Stanford University, Stanford California, 1974. (Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania.) (a)
- Erman, Lee D., (Ed.). Contributed Papers, IEEE Symposium on Speech Recognition. Carnegie-Mellon University, Pittsburgh, Pennsylvania, 15-19 April 1974. IEEE, New York, 1974, 250-254. (b)
- Fennell, Richard D., and Lesser, Victor R. Parallelism in AI Problem Solving: A Case Study of Hearsay II. Technical Report, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 1975.
- Fillmore, Charles J. The Case for Case. In: Universals in Linguistic Theory. Edited by Emmon Bach and Robert T. Harms. Holt, Rinehart and Winston, 1968. Pp. 1-88.
- Freedle, Roy O. Language Users as Fallible Information-Processors: Implications for Measuring and Modeling Comprehension. In: Language Comprehension and the Acquisition of Knowledge. Edited by John B. Carroll and Roy O. Freedle. Winston, Washington, D.C., 1972, 169-209.
- Goodman, R. Gary. Analysis of Languages for Man-Machine Communication. Ph.D. Thesis, Stanford University, Stanford, California, 1976. (Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania.)
- Goodman, R. Gary, Lowerre, Bruce T., and Reddy, D. Raj. Effects of Branching Factor and Vocabulary Size on Performance (Abstract). In: Speech Understanding Systems: Summary of Results of the Five-Year Research Effort. By Reddy, D. Raj, et al. Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania, September 1976.
- Green, C. Cordell. The Application of Theorem Proving to Question-Answering Systems. Technical Report No. CS 138, Artificial Intelligence Group, Stanford Research Institute, Menlo Park, California, and Artificial Intelligence Project, Stanford University, Stanford, California, June 1969.

- Grosz, Barbara J. The Representation and Use of Focus in Dialog Understanding. Ph.D. Thesis, University of California, Berkeley, California, 1977.
- Halliday, Michael A., and Hasan, Ruqaiya. Cohesion in English. London, Longman, 1976.
- Hankamer, Jorge, and Sag, Ivar. Deep and Surface Anaphora. Linguistic Inquiry, 1976, 7, 390-428.
- Hart, Peter E. Progress on a Computer Based Consultant. Technical Note 99, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, January 1975.
- Hart, Peter E., Nilsson, Nils J., and Raphael, Bertram. A Formal Basis for the Heuristic Determination of Minimal Cost Paths. IEEE Transactions on Systems Science and Cybernetics, 1968, SSC-4, 100-107.
- Hayes-Roth, Frederick, and Mostow, David J. An Automatically Compilable Recognition Network for Structured Patterns. Advance Papers, International Joint Conference on Artificial Intelligence, Tbilisi, Georgian SSR, 3-8 September 1975.
- Heidorn, George E. Augmented Phrase Structure Grammars. In: Theoretical Issues in Natural Language Processing. Edited by Roger C. Schank and Bonnie Nash-Webber. Center for Applied Linguistics, Arlington, Virginia, 1975, 1-5.
- Hendrix, Gary G. Expanding the Utility of Semantic Networks Through Partitioning. Advance Papers, International Joint Conference on Artificial Intelligence, Tbilisi, Georgian SSR, 3-8 September 1975, 115-121. (a)
- Hendrix, Gary G. Partitioned Networks for the Mathematical Modeling of Natural Language Semantics. Technical Report NL-28, Department of Computer Sciences, University of Texas, Austin, Texas, 1975. (b)
- Hendrix, Gary G. Semantic Processing for Speech Understanding. American Journal of Computational Linguistics, 1975, 4, Microfiche 34. (c)
- Hintikka, Jaakko. Quantifiers vs. Quantification Theory. Linguistic Inquiry, 1974, 5, 153-177.
- Hobbs, Jerry R. A Metalanguage for Expressing Grammatical Restrictions in Nodal Spans Parsing of Natural Language. Courant Computer Science Report No. 2, New York University, New York, 1974.

- Hobbs, Jerry R. Pronoun Resolution. Research Report 76-1, Department of Computer Sciences, City University of New York, New York, August 1976.
- Hornby, A. S., Gatenby, E. V., and Wakefield, H.. The Advanced Learner's Dictionary of Current English. Oxford Press, London, 1948.
- Hornby, A. S. A Guide to Patterns and Usage in English. Oxford Press, London, 1954.
- Irons, E. T. A Syntax Directed Compiler for ALGOL 60. Communications of the ACM, 1961, 4, 51-55.
- Jazayeri, Mehdi, Ogden, William F., and Rounds, William C. The Intrinsically Exponential Complexity of the Circularity Problem for Attribute Grammars. Communications of the ACM, 1975, 18, 697-706.
- Kaplan, Ronald M. A General Syntactic Processor. In: Natural Language Processing. Edited by Randall Rustin. Algorithmics Press, New York, 1973. Pp. 193-241.
- Kaplan, Ronald M. A Multi-Processing Approach to Natural Language. Proceedings, National Computer Conference, New York, New York, 4-8 June 1973. Volume 42. AFIPS Press, New Jersey, 1973, 435-440. (b)
- Kay, Martin. Experiments With a Powerful Parser. RM-5452-PR, The RAND Corporation, Santa Monica, California, 1967.
- Kay, Martin. The Mind System. In: Natural Language Processing. Edited by Randall Rustin. Algorithmics Press, New York, 1973. Pp. 153-188.
- Knuth, Donald E. Semantics of Context-Free Languages. Mathematical Systems Theory, 1968, 2, 127-145.
- Landsbergen, S. P. Jan. Syntax and Formal Semantics of English in PHLIQA1. In: COLING 76, Preprints of the 6th International Conference on Computational Linguistics, Ottawa, Ontario, Canada, 28 June - 2 July 1976. No. 21.
- Lesser, Victor R., Fennell, Richard D., Erman, Lee D., and Reddy, D. Raj. Organization of the Hearsay II Speech Understanding System. IEEE Transactions on Acoustics, Speech, and Signal Processing, 1975, ASSP-23, 11-24.
- Lowerre, Bruce T. The HARP Speech Recognition System. Ph.D. Thesis, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 1976.
- Malhotra, Ashok. Design Requirements for a Knowledge-Based English Language System for Management: An Experimental Analysis. Ph.D.

- Thesis, Sloan School of Management, Massachusetts Institute of Technology, February 1975.
- Miller, Perry L. A Locally Organized Parser for Spoken Input. Technical Report 503, Lincoln Laboratory, Massachusetts Institute of Technology, Lexington, Massachusetts, May 1973.
- Minsky, Marvin. A Framework for Representing Knowledge. AI Memo 306, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1974.
- Newell, Allen. Production Systems: Models of Control Structures. In: Visual Information Processing. Edited by W. C. Chase. Academic Press, New York, 1973. Pp. 463-526.
- Newell, Allen. A Tutorial on Speech Understanding Systems. In: Speech Recognition: Invited Papers of the 1974 IEEE Symposium. Edited by D. R. Reddy. Academic Press, New York, 1975. Pp. 3-54.
- Newell, Allen, and Simon, Herbert A. Computer Science as Empirical Inquiry: Symbols and Search. 1975 ACM Turing Award Lecture. Communications of the ACM, 1976, 19, 113-126.
- Newell, Allen, et al. Speech Understanding Systems. North-Holland Publishing Company, Amsterdam, 1973.
- Nilsson, Nils J., et al. Artificial Intelligence -- Research and Applications. Annual Report, Project 3805, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, May 1975.
- Norman, Donald A., Rumelhart, David E., and the LNR Research Group. Explorations in Cognition. Freeman, San Francisco, 1975.
- Olson, David R. Language and Thought: Aspects of a Cognitive Theory of Semantics. Psychological Review, 1970, 77, 257-273.
- Paxton, William H. A Best-First Parser. IEEE Transactions on Acoustics, Speech, and Signal Processing, 1975, ASSP-23, 426-432.
- Paxton, William H. A Framework for Language Understanding. In: COLING 76, Preprints of the 6th International Conference on Computational Linguistics, Ottawa, Ontario, Canada, 28 June - 2 July 1976. No. 14. (a)
- Paxton, William H. Experiments in Speech Understanding System Control. In Proceedings of the First CSCSI/SCEIO National Conference, Vancouver, British Columbia, Canada, 25-27 August 1976. (b)

- Paxton, William H., and Robinson, Ann E. A Parser for a Speech Understanding System. Advance Papers, International Joint Conference on Artificial Intelligence, Stanford, California, 20-23 August 1973. Stanford Research Institute, Menlo Park, California, 1973, 216-222.
- Paxton, William H., and Robinson, Ann E. System Integration and Control in a Speech Understanding System. American Journal of Computational Linguistics, 1975, 4, Microfiche 34.
- Petrick, Stanley R. Semantic Interpretation in the Request System. In: Computational and Mathematical Linguistics, Proceedings of the International Conference on Computational Linguistics, Volume I. Edited by Antonio Zampolli. Casa Editrice Leo S. Olschki, Firenze, 1973.
- Petrick, Stanley R. On Natural Language Based Computer Systems. IBM Journal of Research and Development, 1976, 20, 314-325.
- Pratt, Vaughn R. LINGOL - A Progress Report. Advance Papers, International Joint Conference on Artificial Intelligence, Tbilisi, Georgian SSR, 3-8 September 1975, 422-428.
- Reboh, Rene, and Sacerdoti, Earl. A Preliminary QLISP Manual. Technical Note 81, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, August 1973.
- Reddy, D. Raj. Speech Recognition by Machine: A Review. Proceedings of the IEEE, 1976, 64, 501-531.
- Reddy, D. Raj, et al. Speech Understanding Systems: Summary of Results of the Five-Year Research Effort. Department of Computer Science. Carnegie-Mellon University, Pittsburgh, Pennsylvania, September 1976.
- Ritea, H. Barry. A Voice-Controlled Data Management System. Contributed Papers, IEEE Symposium on Speech Recognition. Carnegie-Mellon University, Pittsburgh, Pennsylvania, 15-19 April 1974. Edited by Lee D. Erman. IEEE, New York, 1974, 28-31.
- Ritea, H. Barry. Automatic Speech Understanding Systems. Proceedings of the 11th Annual IEEE Computer Society Conference, Washington, D.C., September 1975.
- Robinson, Jane J. Performance Grammars. In: Speech Recognition: Invited Papers of the 1974 IEEE Symposium. Edited by D. Raj Reddy. Academic Press, New York, 1975. Pp. 401-427. (a)
- Robinson, Jane J. A Tuneable Performance Grammar. American Journal of Computational Linguistics, 1975, 4, Microfiche 34. (b)

- Rumelhart, David E., and Norman, Donald A. Active Semantic Networks as a Model of Human Memory. Advance Papers, International Joint Conference on Artificial Intelligence, Stanford, California, 20-23 August 1973. Stanford Research Institute, Menlo Park, California, 1973, 450-457.
- Sager, Naomi, and Grishman, Ralph. The Restriction Language for Computer Grammars. Communications of the ACM, 1975, 18, 390-400.
- Scha, Remko J. H. Semantic Types in PHLIQA1. In: COLING 76, Preprints of the 6th International Conference on Computational Linguistics, Ottawa, Ontario, Canada, 28 June - 2 July 1976. No. 10.
- Schank, Roger C. Identification of Conceptualizations Underlying Natural Language. In: Computer Models of Thought and Language. Edited by Roger C. Schank and Kenneth M. Colby. W. H. Freeman, San Francisco, 1973, 187-247.
- Shapiro, Stuart C. A Net Structure for Semantic Information Storage, Deduction and Retrieval. Advance Papers, Second International Joint Conference on Artificial Intelligence, London, England, 1-2 September 1971. The British Computer Society, London, England, 1971. Pp. 512-523.
- Silva, Georgette. SDC-SRI Protocol Gathering Experiments and Computer Analysis of Dialog. SUR Note 141, System Development Corporation, Santa Monica, California, October 1975.
- Simmons, Robert F. Semantic Networks: Their Computation and Use for Understanding English Sentences. In: Computer Models of Thought and Language. Edited by Roger C. Schank and Kenneth M. Colby. Freeman, San Francisco, 1973. Pp. 63-113.
- Slocum, Jonathan. Speech Generation from Semantic Nets. American Journal of Computational Linguistics, 1975, 4, Microfiche 33.
- Sowa, John F. Conceptual Graphs for a Data Base Interface. IBM Journal of Research and Development, 1976, 20, 336-357.
- Teitelman, Warren. INTERLISP Reference Manual. XEROX Palo Alto Research Center, Palo Alto, California, 1975.
- Thorne, James P., Bratley, Paul, and Dewar, Haimish. The Syntactic Analysis of English by Machine. In: Machine Intelligence 3. Edited by Donald Michie. Edinburgh University Press, Edinburgh, 1968.
- Uppsala University. Interlisp/360 and /370 User Reference Manual. Uppsala University, Uppsala, Sweden, 1975.

- Walker, Donald E. Speech Understanding Research. Annual Report, Project 1526, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, February 1973. (a)
- Walker, Donald E. Speech Understanding Through Syntactic and Semantic Analysis. Advance Papers, International Joint Conference on Artificial Intelligence, Stanford, California, 20-23 August 1973. Stanford Research Institute, Menlo Park, California, 1973, 208-215. (b)
- Walker, Donald E. Speech Understanding Research. Annual Report, Project 1526, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, May 1974.
- Walker, Donald E. The SRI Speech Understanding System. IEEE Transactions on Acoustics, Speech, and Signal Processing, 1975, ASSP-23, 397-416.
- Walker, Donald E., et al. Speech Understanding Research. Annual Report, Project 3804, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, June 1975.
- Winer, B. J. Statistical Principles in Experimental Design. Second Edition. McGraw-Hill, New York, 1971.
- Winograd, Terry. Procedures as a Representation for Data in a Computer Program for Understanding Natural Language. Report MAC-TR-84, Project MAC, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1971. [Published as Understanding Natural Language, Academic Press, New York, 1972.]
- Winograd, Terry. Frame Representations and the Declarative/Procedural Controversy. In: Representation and Understanding. Edited by Daniel G. Bobrow and Allen M. Collins. Academic Press, New York, 1975. Pp. 185-210.
- Woods, William A. Transition Network Grammars for Natural Language Analysis. Communications of the ACM, 1970, 13, 591-606.
- Woods, William A. What's in a Link: Foundations for Semantic Networks. In: Representation and Understanding. Edited by Daniel G. Bobrow and Allen M. Collins. Academic Press, New York, 1975. Pp. 35-82.
- Woods, William A., Kaplan, Ronald M., and Nash-Webber, Bonnie. The Lunar Sciences Natural Language Information System. Final Report. BBN Report 2378, Bolt Beranek and Newman, Cambridge, Massachusetts, 1972.
- Woods, William A., and Makhoul, John. Mechanical Inference Problems in Continuous Speech Understanding. Artificial Intelligence, 1974, 5, 73-91.

Woods, William A., et al. Speech Understanding Systems. Quarterly Technical Progress Report No. 1. BBN Report 3018, Bolt Beranek and Newman, Cambridge, Massachusetts, February 1975. (a)

Woods, William A., et al. Speech Understanding Systems. Annual Technical Progress Report. BBN Report 3188, Bolt Beranek and Newman, Cambridge, Massachusetts, October 1975. (b)

Woods, William A., et al. Speech Understanding Systems. Quarterly Technical Progress Report No. 5. BBN Report 3240, Bolt Beranek and Newman, Cambridge, Massachusetts, January 1976. (a)

Woods, William A., et al. Speech Understanding Systems. Quarterly Technical Progress Report No. 6. BBN Report 3303, Bolt Beranek and Newman, Cambridge, Massachusetts, April 1976. (b)

XV SRI SPEECH UNDERSTANDING RESEARCH PUBLICATIONS AND REPORTS

- Becker, Richard, and Poza, Fausto. Acoustic Processing in the SRI Speech Understanding System. IEEE Transactions on Acoustics, Speech and Signal Processing, 1975, ASSP-23, 416-426.
- Deutsch, Barbara G. The Structure of Task-Oriented Dialogs. Contributed Papers, IEEE Symposium on Speech Recognition, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 15-19 April 1974. IEEE, New York, 1974, 250-254. [Technical Note 90, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, April 1974.]
- Deutsch, Barbara G. Establishing Context in Task-Oriented Dialogs. American Journal of Computational Linguistics, 1975, Microfiche 35. [Technical Note 114, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, September 1975.]
- Hendrix, Gary G. Expanding the Utility of Semantic Networks Through Partitioning. Advance Papers of the Fourth International Joint Conference on Artificial Intelligence, Tbilisi, Georgia, USSR, 3-8 September 1975, 115-121. [Technical Note 105, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, June 1975.]
- Hendrix, Gary G. Semantic Processing for Speech Understanding. American Journal of Computational Linguistics, 1975, Microfiche 34. [Technical Note 113, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, September 1975.]
- Paxton, William H. A Best-First Parser. IEEE Transactions on Acoustics, Speech and Signal Processing, 1975, ASSP-23, 426-432. [Contributed Papers, IEEE Symposium on Speech Recognition, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 15-19 April 1974. IEEE, New York, 218-225.] [Technical Note 92, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, April 1974.]
- Paxton, William H., and Robinson, Ann E. A Parser for a Speech Understanding System. Advance Papers, International Joint Conference on Artificial Intelligence, Stanford, California, 20-23 August 1973. Stanford Research Institute, Menlo Park, California, 1973, 216-222. [Technical Note 79, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, June 1973.]

Paxton, William H., and Robinson, Ann E. System Integration and Control in a Speech Understanding System. American Journal of Computational Linguistics, 1975, Microfiche 34. [Technical Note 111, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, September 1975.]

Paxton, William H. A Framework for Language Understanding. In COLING 76, Preprints of the 6th International Conference on Computational Linguistics, Ottawa, Ontario, Canada, 28 June - 2 July 1976. [Technical Note 131, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, June 1976.]

Paxton, William H. Experiments in Speech Understanding System Control. In Proceedings of the First CSCSI/SCEIO National Conference, Vancouver, British Columbia, Canada, 25-27 August 1976. [Technical Note 134, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, August 1976.]

Robinson, Jane J. Performance Grammars. In Speech Recognition: Invited Papers of the 1974 IEEE Symposium. Edited by Raj Reddy. Academic Press, New York, 1975. Pp. 401-427. [Invited Papers, IEEE Symposium on Speech Recognition, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 15-19 April 1974.] [Technical Note 97, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, April 1974.]

Robinson, Jane J. A Tuneable Performance Grammar. American Journal of Computational Linguistics, 1975, Microfiche 34. [Technical Note 112, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, September 1975.]

Slocum, Jonathan. Speech Generation from Semantic Nets. American Journal of Computational Linguistics, 1975, Microfiche 33. [Technical Note 115, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, September 1975.]

Walker, Donald E. Speech Understanding Research. Annual Technical Report, Project 1526, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, February 1973.

Walker, Donald E. Speech Understanding Through Syntactic and Semantic Analysis. IEEE Transactions on Computers, 1976, C-25, 432-439. [Advance Papers, International Joint Conference on Artificial Intelligence, Stanford, California, 20-23 August 1973. Stanford Research Institute, Menlo Park, California, 1973, 208-215.] [Technical Note 80, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, June 1973.]

Walker, Donald E. Speech Understanding, Computational Linguistics, and Artificial Intelligence. In Computational and Mathematical

Linguistics, Proceedings of the International Conference on Computational Linguistics, Volume I. Edited by Antonio Zampolli. Casa Editrice Leo S. Olschki, Firenze, 1973. Pp. 725-740. [Technical Note 85, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, August 1973.]

Walker, Donald E. Speech Understanding Research. Annual Technical Report, Project 1526, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, May 1974.

Walker, Donald E. The SRI Speech Understanding System. IEEE Transactions on Acoustics, Speech and Signal Processing, 1975, ASSP-23, 397-416. [Contributed Papers, IEEE Symposium on Speech Recognition, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 15-19 April 1974. IEEE, New York, 32-37.] [Technical Note 91, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, April 1974.]

Walker, Donald E. Progress in Speech Understanding Research at SRI. Proceedings of the Fourth International Congress of Applied Linguistics, Stuttgart, German Federal Republic, 25 - 30 August 1975. Edited by Gerhard Nickel. Hochschul Verlag, Stuttgart, 1976. [Technical Note 110, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, August 1975.]

Walker, Donald E., et al. Speech Understanding Research. Annual Technical Report, Project 3804, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, June 1975.

Walker, Donald E. Speech Understanding Research. Semiannual Technical Report, Project 4762, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, June 1976.

Walker, Donald E. (Ed.) Speech Understanding Research. Final Technical Report, Project 4762, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, October 1976.